

Probabilistic Programming

KI 2014

Angelika Kimmig

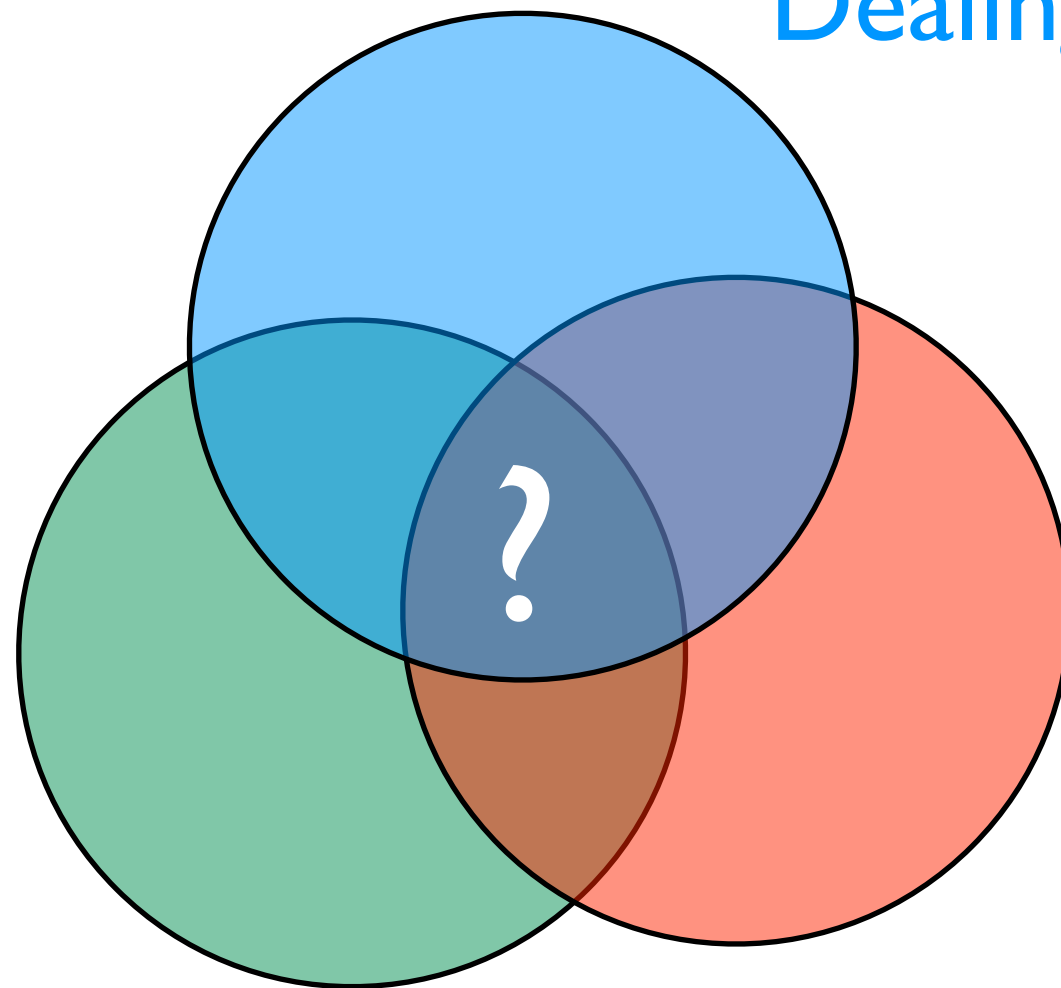
angelika.kimmig@cs.kuleuven.be



A key question in AI:

Dealing with uncertainty

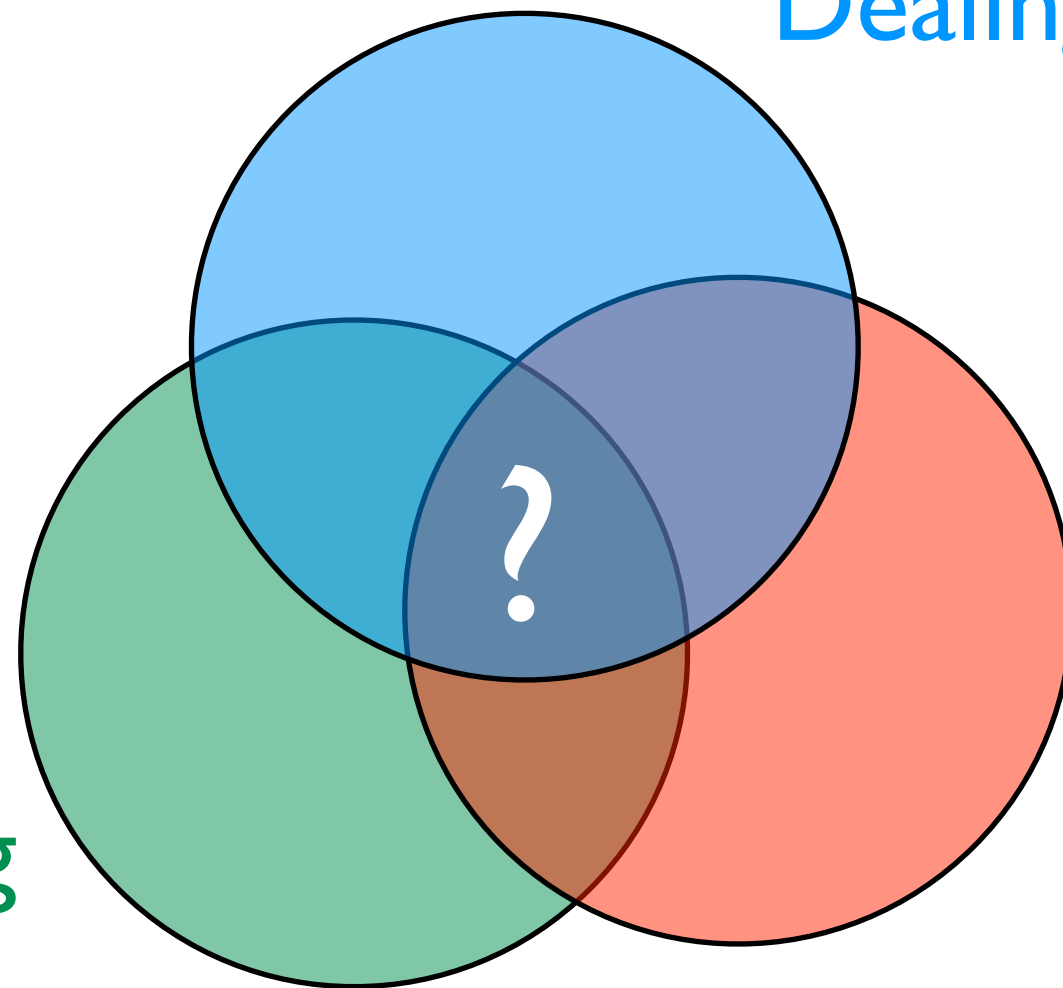
Reasoning with
relational data



Learning

A key question in AI:

Dealing with uncertainty



Learning

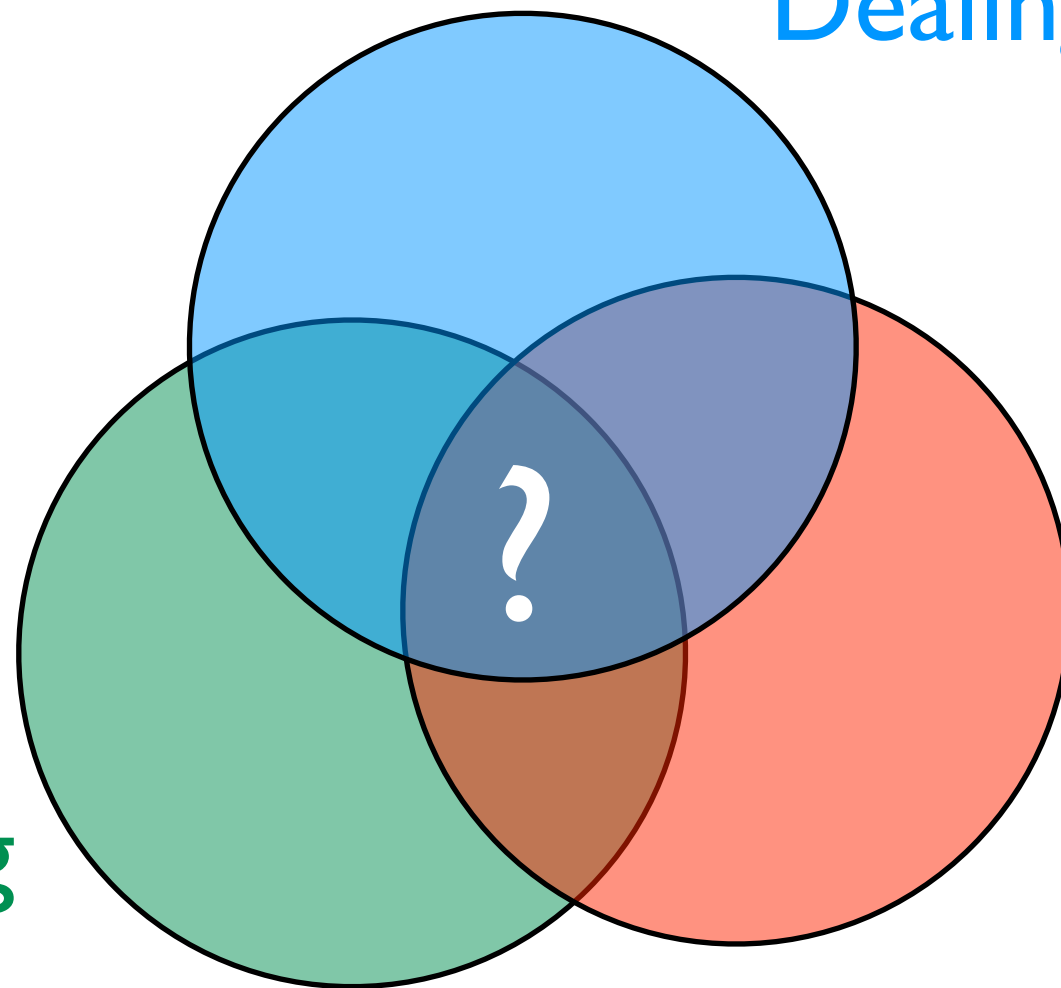
Reasoning with
relational data

- logic
- databases
- programming
- ...

A key question in AI:

Reasoning with
relational data

- logic
- databases
- programming
- ...



Dealing with uncertainty

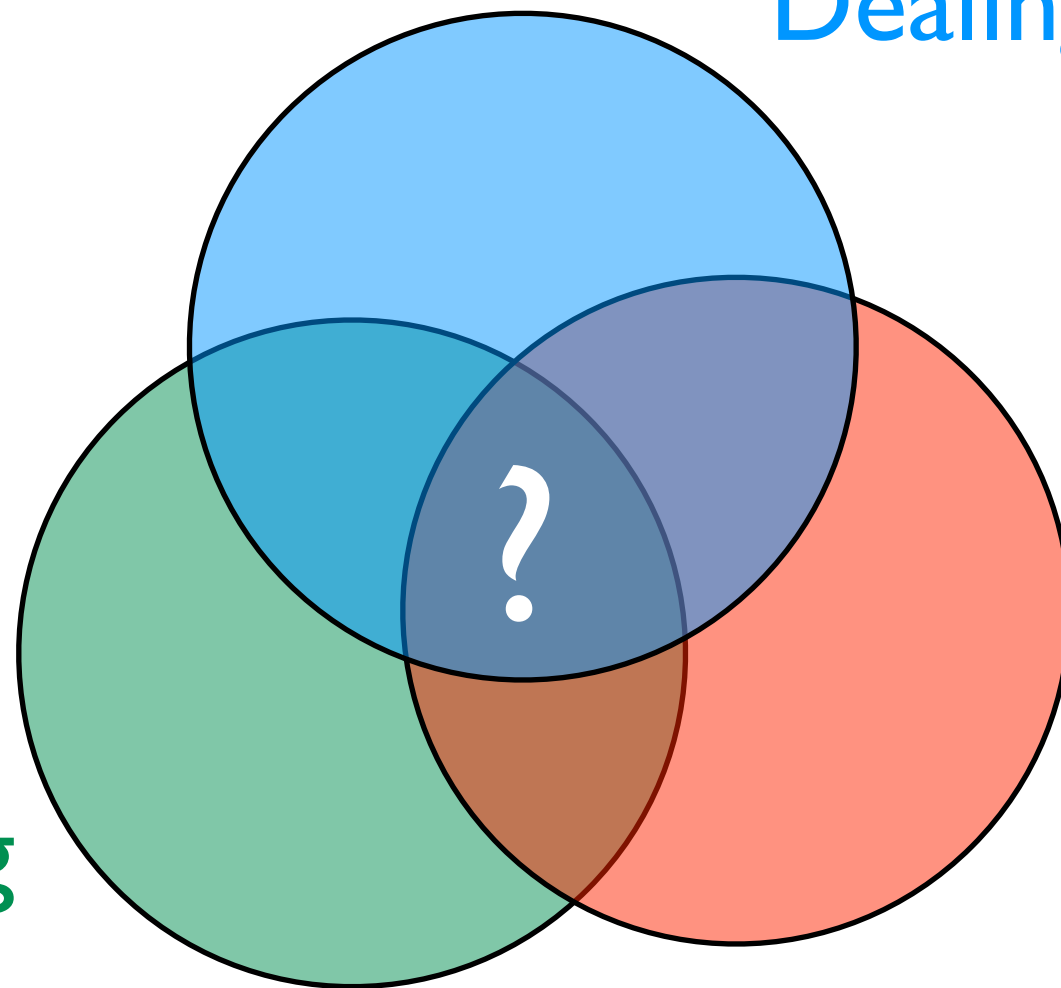
- probability theory
- graphical models
- ...

Learning

A key question in AI:

Reasoning with
relational data

- logic
- databases
- programming
- ...



Dealing with uncertainty

- probability theory
- graphical models
- ...

Learning

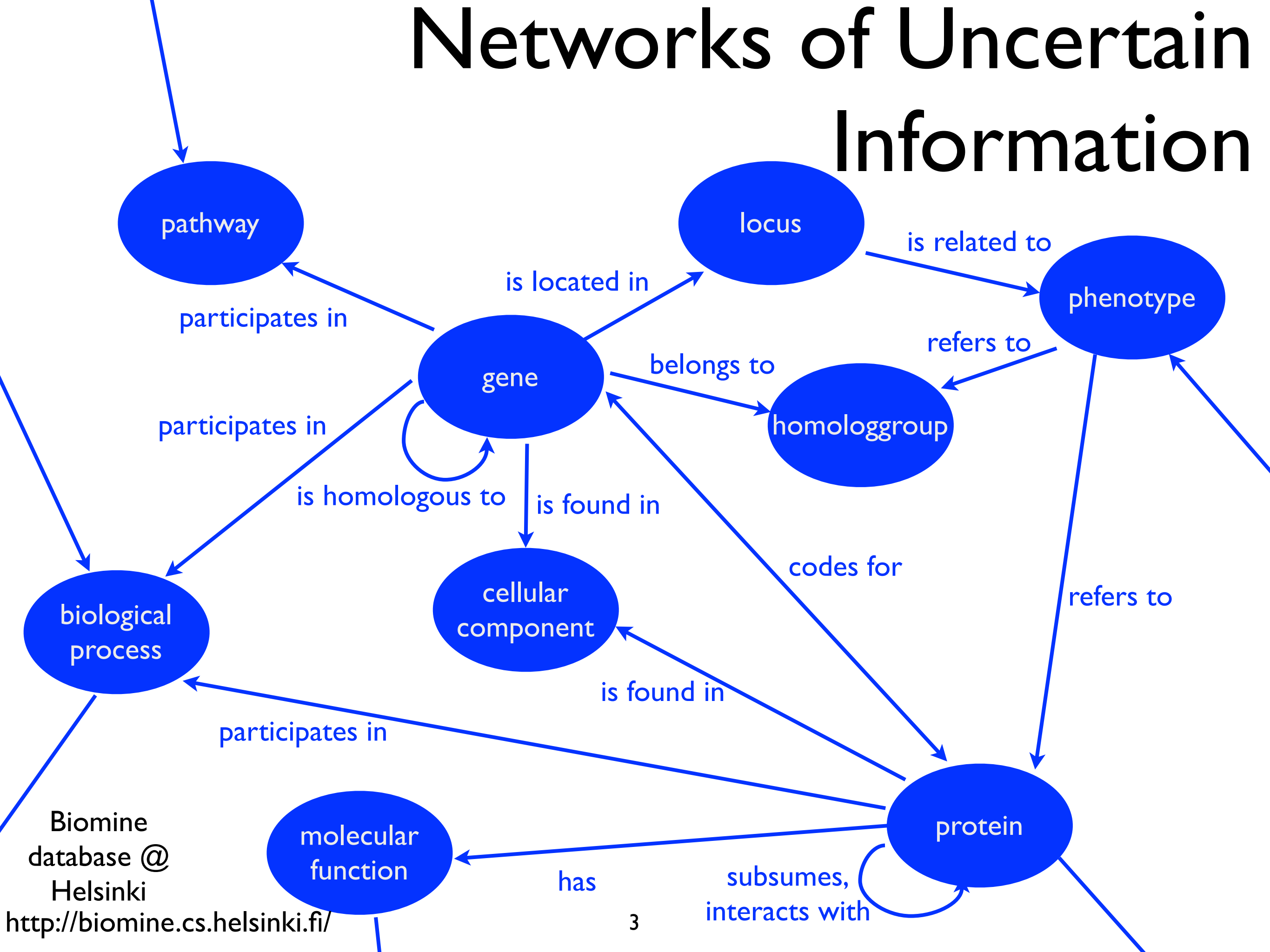
- parameters
- structure

A key question in AI:



Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

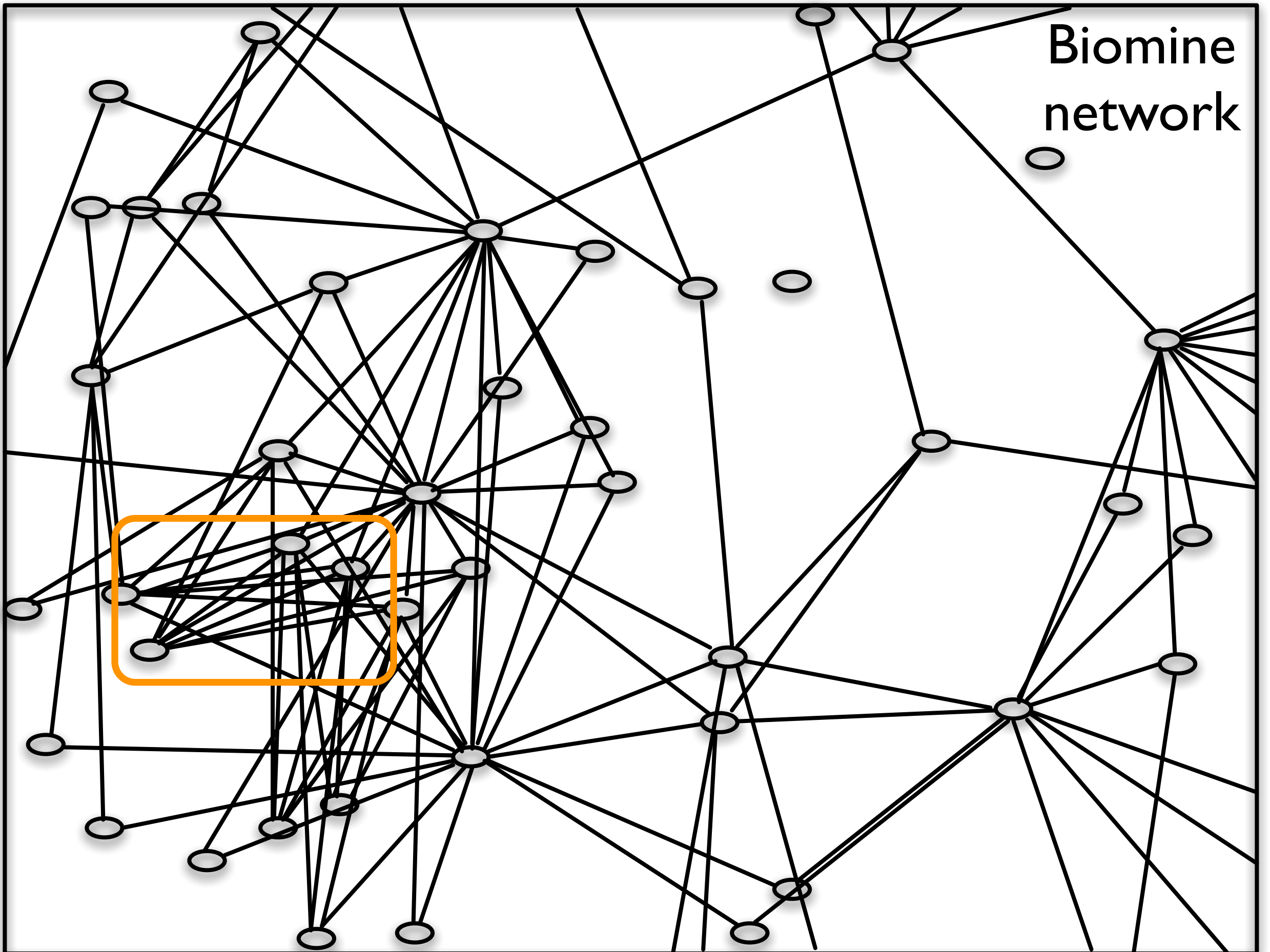
Networks of Uncertain Information



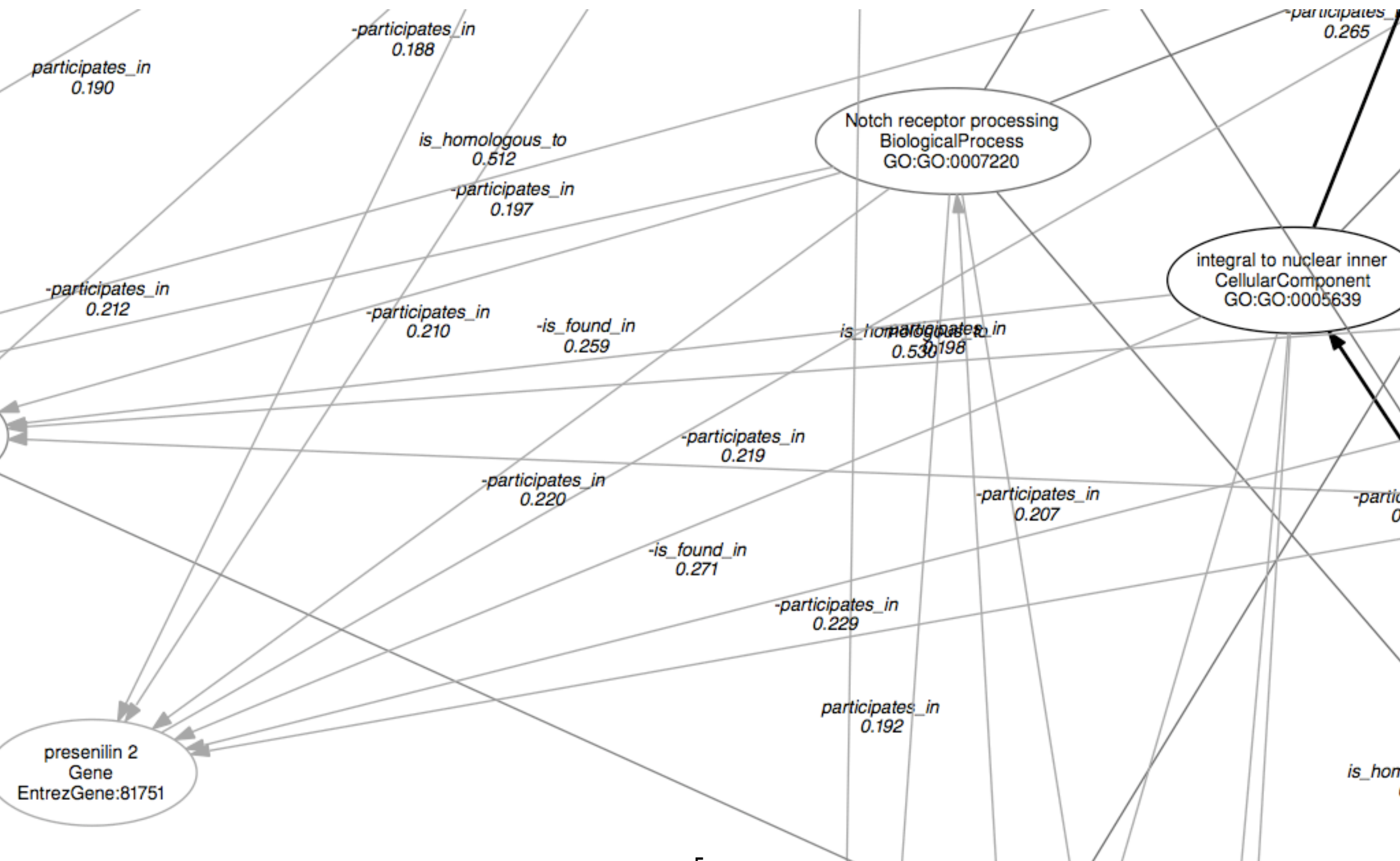
Biomine network



Biomine network



Biomine Network



Biomine Network

BiologicalProcess

-participates_in
0.220

participates_in
0.220

Notch receptor processing
BiologicalProcess
GO:GO:0007220

integral to nuclear inner
CellularComponent
GO:GO:0005639

presenilin 2
Gene
EntrezGene:81751

Gene

Biomine Network

BiologicalProcess

-participates_in
0.220

Notch receptor processing
BiologicalProcess
GO:GO:0007220

- different types of nodes & links
- automatically extracted from text, databases, ...
- probabilities quantifying source reliability, extractor confidence, ...
- similar in other contexts, e.g., linked open data, knowledge graphs, ...

presenilin 2
Gene
EntrezGene:81751

Gene





















Example:

Information Extraction

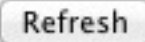

Recently-Learned Facts





















twitter

Refresh

instance	iteration	date learned	confidence
<u>kelly andrews</u> is a <u>female</u>	826	29-mar-2014	98.7  
<u>investment next year</u> is an <u>economic sector</u>	829	10-apr-2014	95.3  
<u>shibenik</u> is a <u>geopolitical entity</u> that is an organization	829	10-apr-2014	97.2  
<u>quality web design work</u> is a <u>character trait</u>	826	29-mar-2014	91.0  
<u>mercedes benz cls by carlsson</u> is an <u>automobile manufacturer</u>	829	10-apr-2014	95.2  
<u>social work</u> is an academic program <u>at the university rutgers university</u>	827	02-apr-2014	93.8  
<u>dante wrote</u> the book <u>the divine comedy</u>	826	29-mar-2014	93.8  
<u>willie aames</u> was <u>born in</u> the city <u>los angeles</u>	831	16-apr-2014	100.0  
<u>kitt peak</u> is a mountain <u>in the state or province arizona</u>	831	16-apr-2014	96.9  
<u>greenwich</u> is a park <u>in the city london</u>	831	16-apr-2014	100.0  

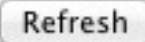
Example: Information Extraction


















Recently-Learned Facts 

instance	iteration	date learned	confidence
<u>kelly andrews</u> is a <u>female</u>	826	29-mar-2014	98.7  
<u>investment next year</u> is an <u>economic sector</u>	829	10-apr-2014	95.3  
<u>shibenik</u> is a <u>geopolitical entity</u> that is an organization	829	10-apr-2014	97.2  
<u>quality web design work</u> is a <u>character trait</u>	826	29-mar-2014	91.0  
<u>mercedes benz cls by carlsson</u> is an <u>automobile manufacturer</u>	829	10-apr-2014	95.2  
<u>social work</u> is an academic program <u>at the university rutgers university</u>	827	02-apr-2014	93.8  
<u>dante wrote</u> the book <u>the divine comedy</u>	826	29-mar-2014	93.8  
<u>willie aames</u> was <u>born in</u> the city <u>los angeles</u>	831	16-apr-2014	100.0  
<u>kitt peak</u> is a mountain <u>in the state or province arizona</u>	831	16-apr-2014	96.9  
<u>greenwich</u> is a park <u>in the city london</u>	831	16-apr-2014	100.0  

↑
instances for many
different relations

Example: Information Extraction

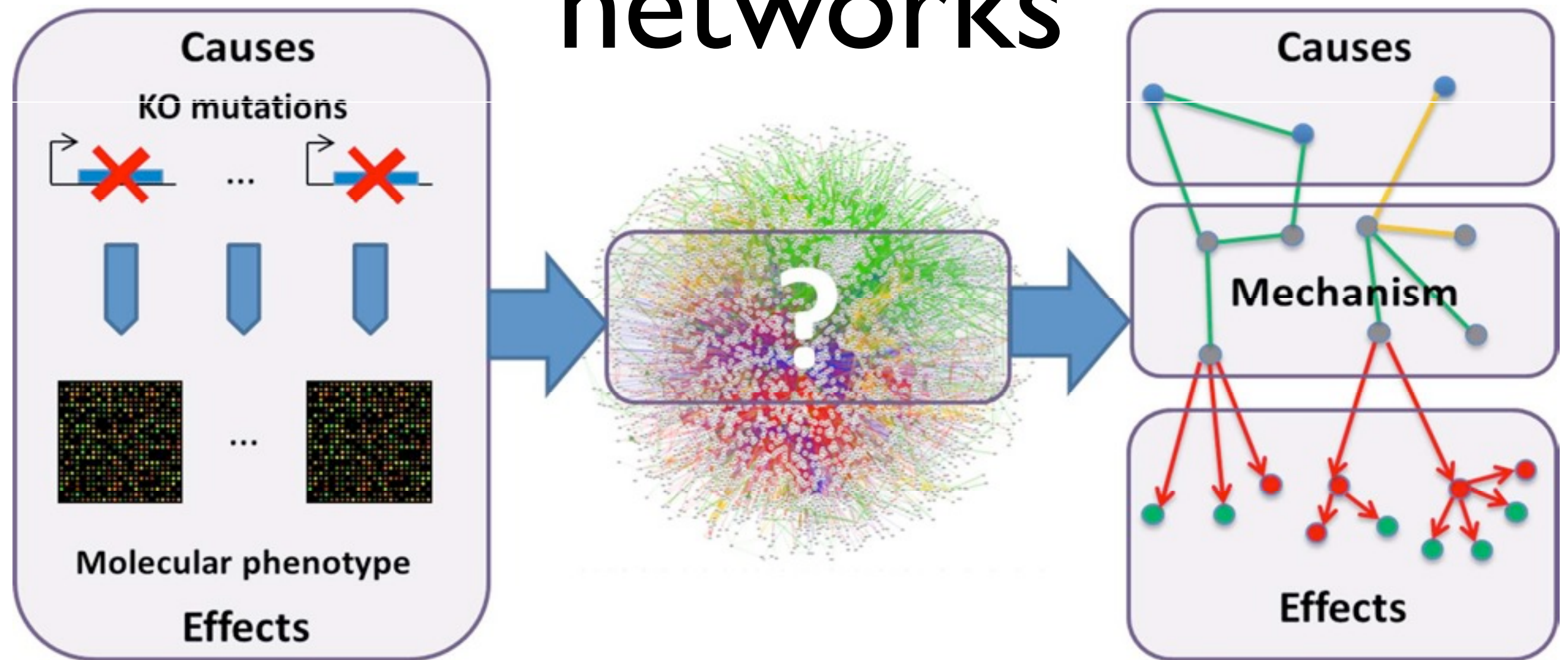
Recently-Learned Facts 

instance	iteration	date learned	confidence
<u>kelly andrews</u> is a <u>female</u>	826	29-mar-2014	98.7  
<u>investment next year</u> is an <u>economic sector</u>	829	10-apr-2014	95.3  
<u>shibenik</u> is a <u>geopolitical entity</u> that is an organization	829	10-apr-2014	97.2  
<u>quality web design work</u> is a <u>character trait</u>	826	29-mar-2014	91.0  
<u>mercedes benz cls by carlsson</u> is an <u>automobile manufacturer</u>	829	10-apr-2014	95.2  
<u>social work</u> is an academic program <u>at the university rutgers university</u>	827	02-apr-2014	93.8  
<u>dante wrote</u> the book <u>the divine comedy</u>	826	29-mar-2014	93.8  
<u>willie aames</u> was <u>born in</u> the city <u>los angeles</u>	831	16-apr-2014	100.0  
<u>kitt peak</u> is a mountain <u>in the state or province arizona</u>	831	16-apr-2014	96.9  
<u>greenwich</u> is a park <u>in the city london</u>	831	16-apr-2014	100.0  

↑
instances for many
different relations

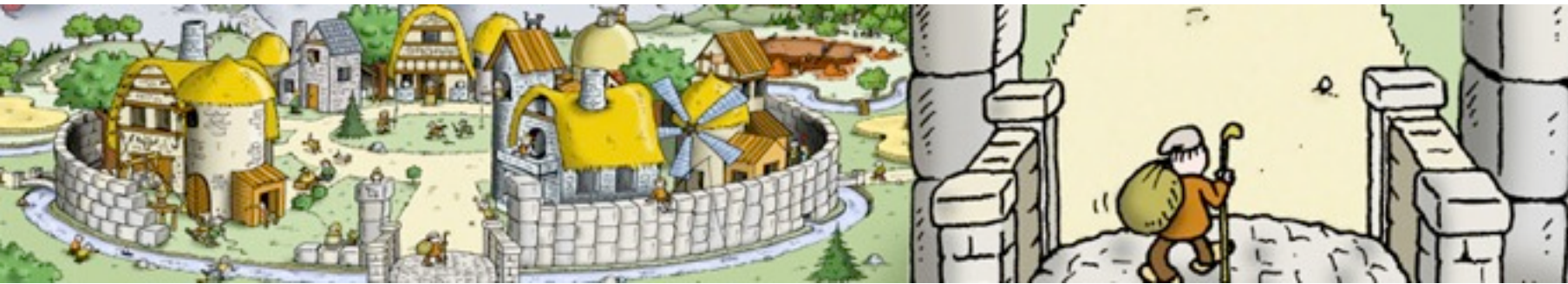
↑
degree of certainty

Molecular interaction networks



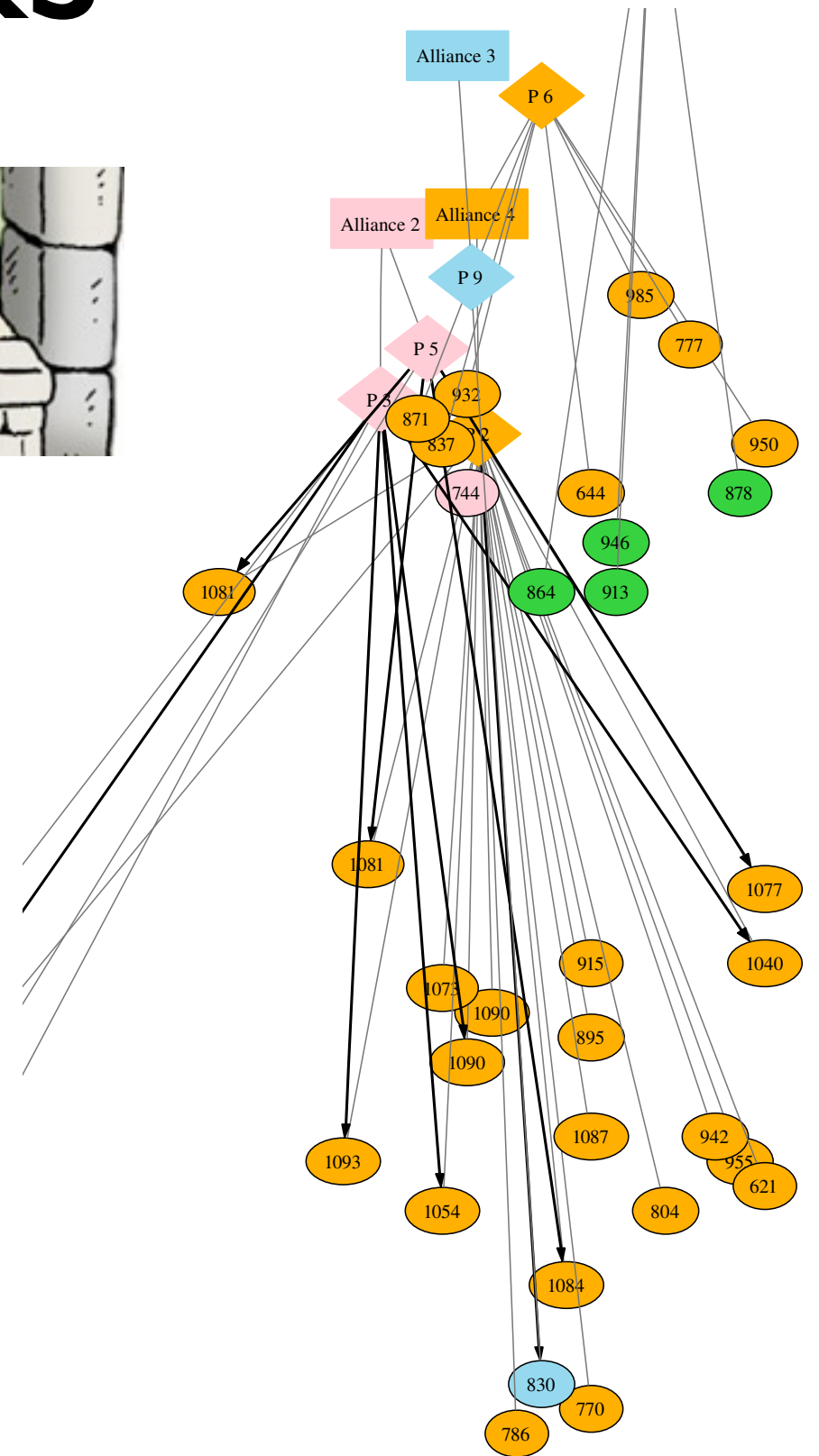
Can we find the mechanism connecting causes to effects?

Dynamic networks

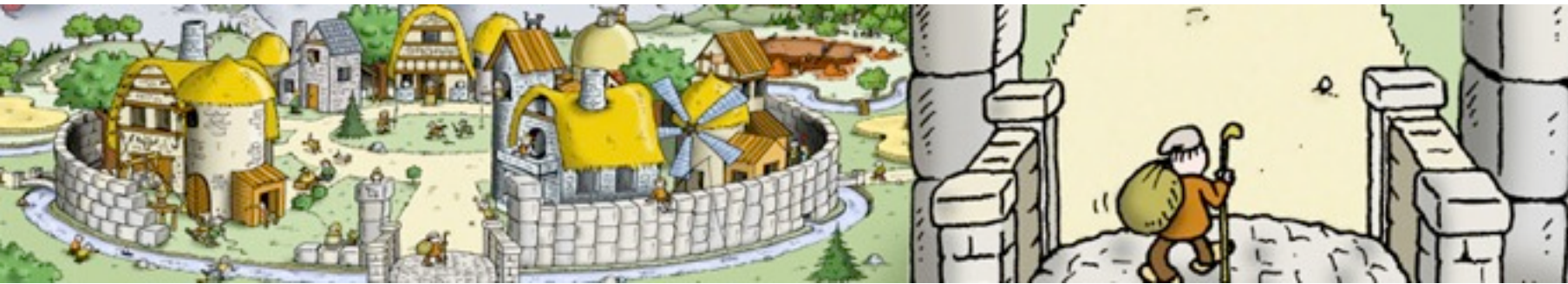


Travian: A massively multiplayer
real-time strategy game

Can we build a model
of this world ?
Can we use it for playing
better ?

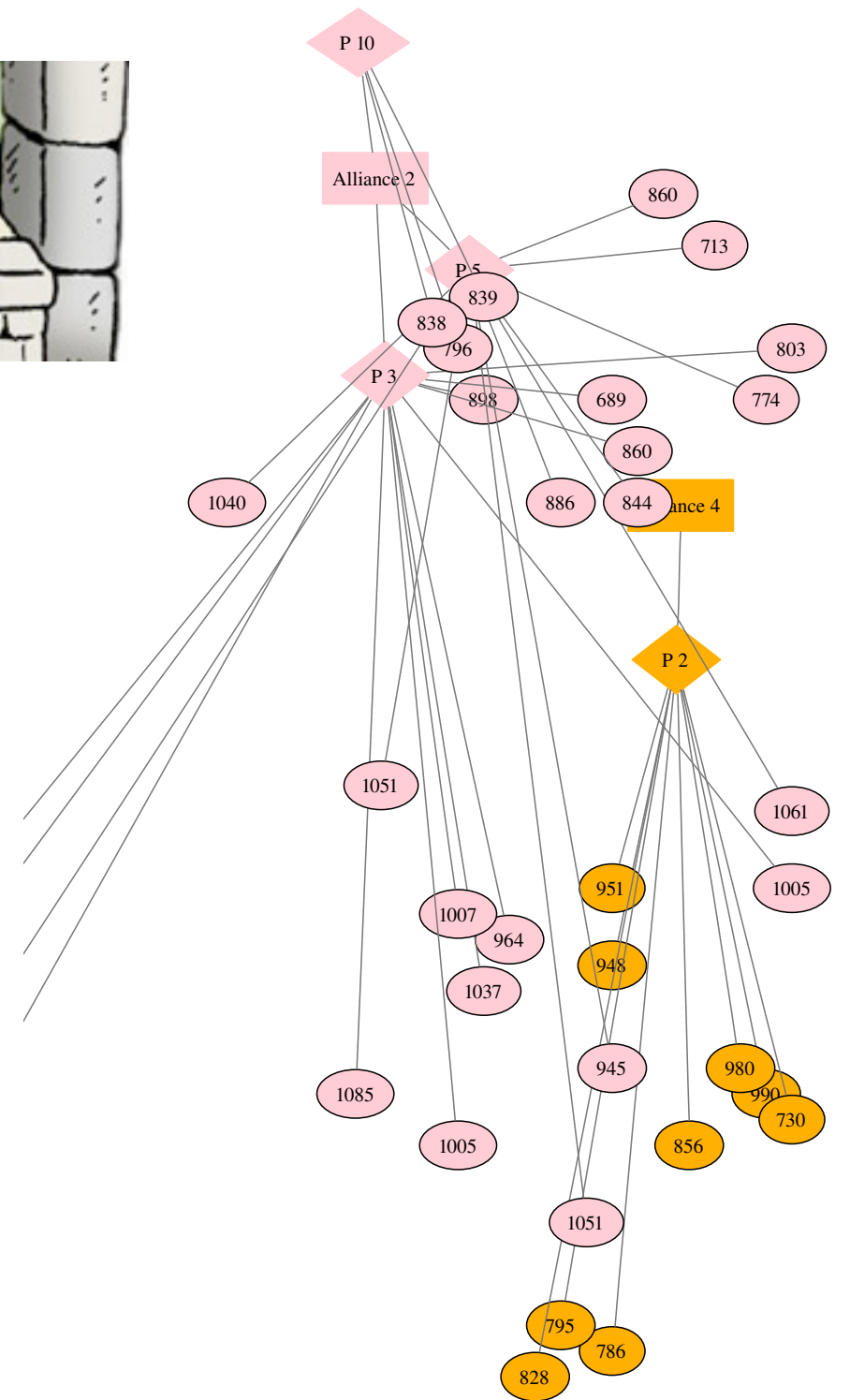


Dynamic networks

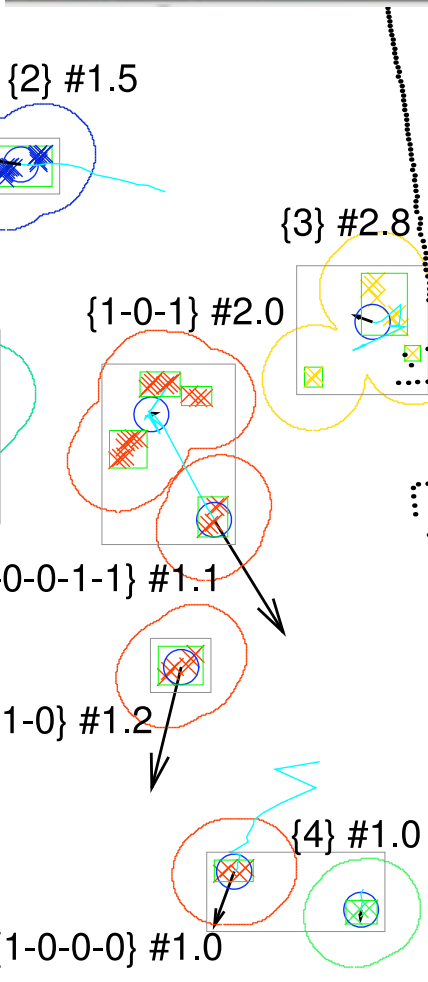


Travian: A massively multiplayer
real-time strategy game

Can we build a model
of this world ?
Can we use it for playing
better ?



Analyzing Video Data



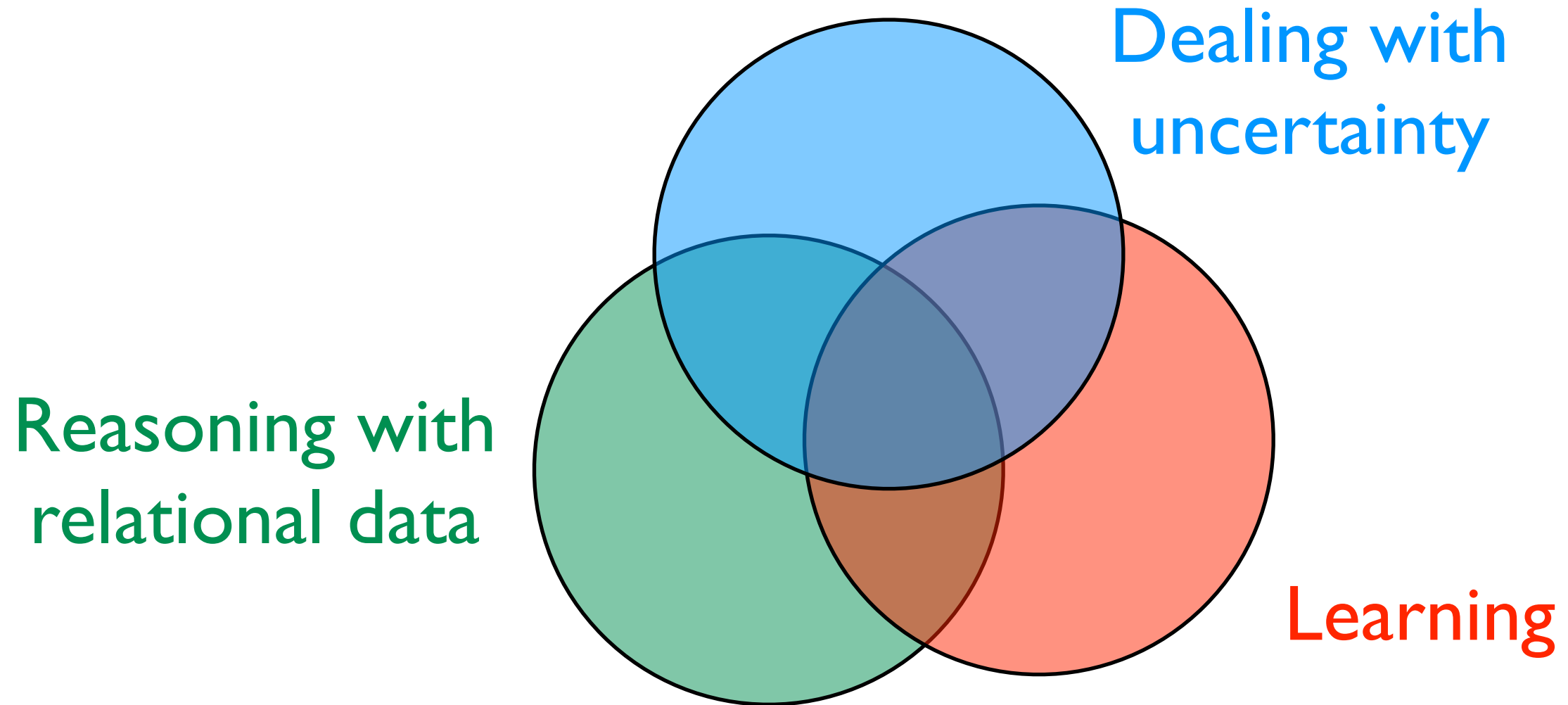
- Track people or objects over time? Even if temporarily hidden?
- Recognize activities?
- Infer object properties?



[Skarlatidis et al, TPLP 14;

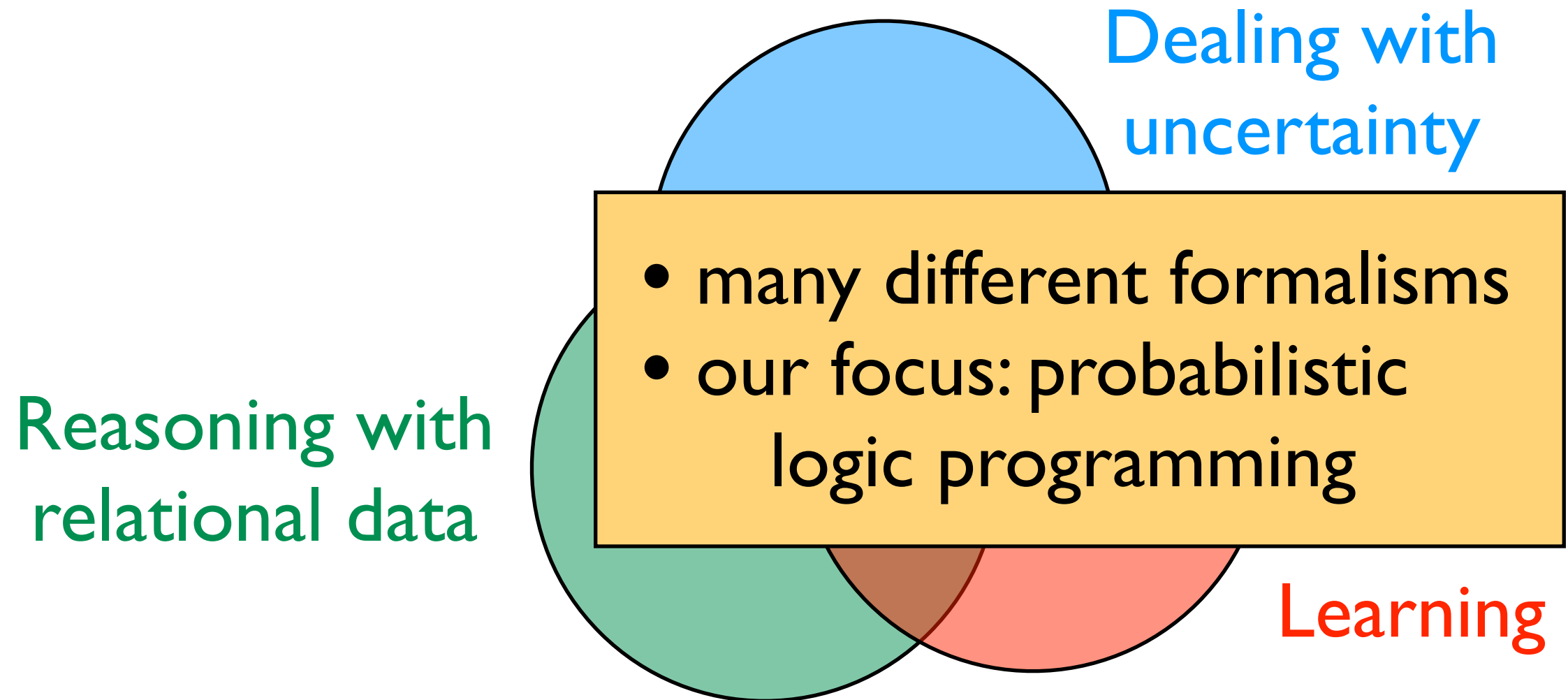
Nitti et al, IROS 13, ICRA 14]

Common theme



Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

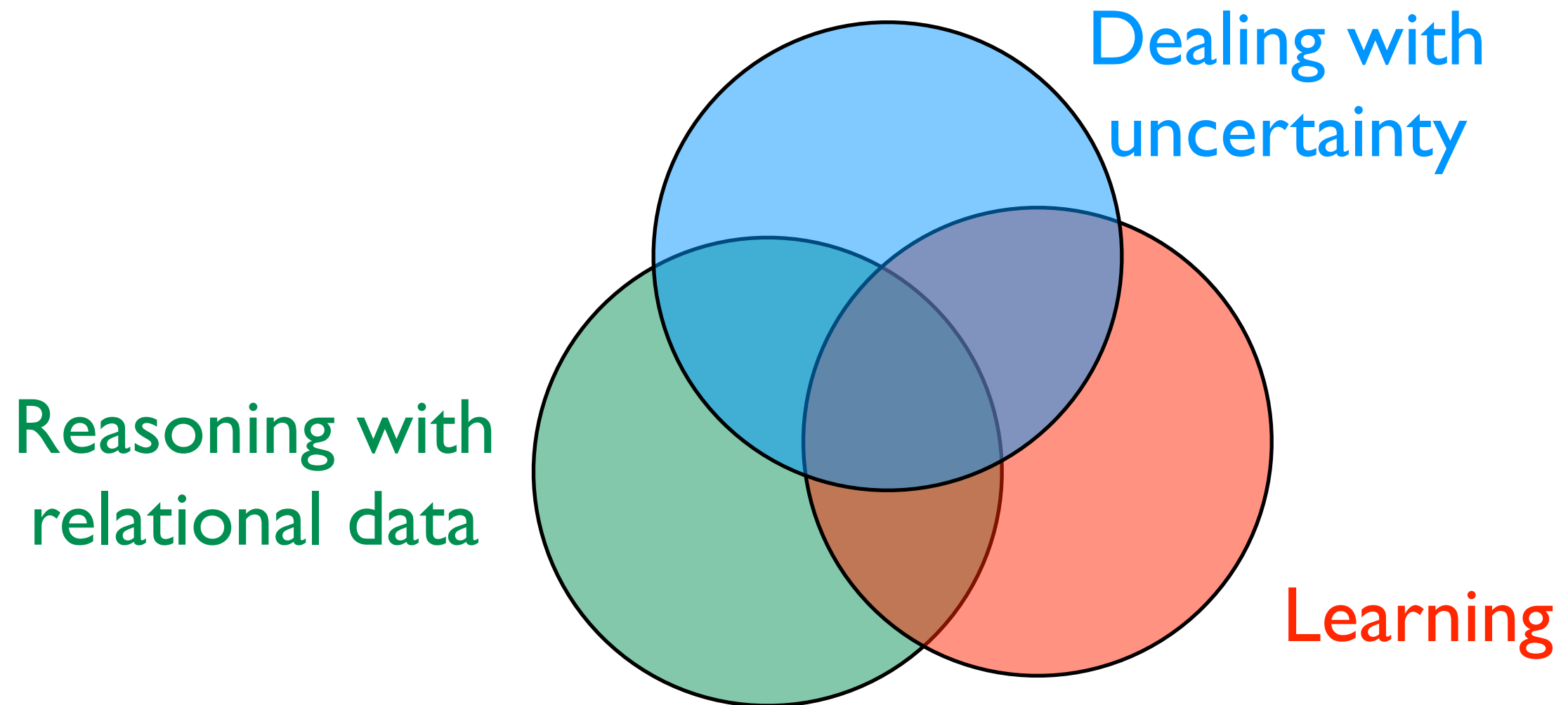
Common theme



Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

ProbLog

probabilistic Prolog



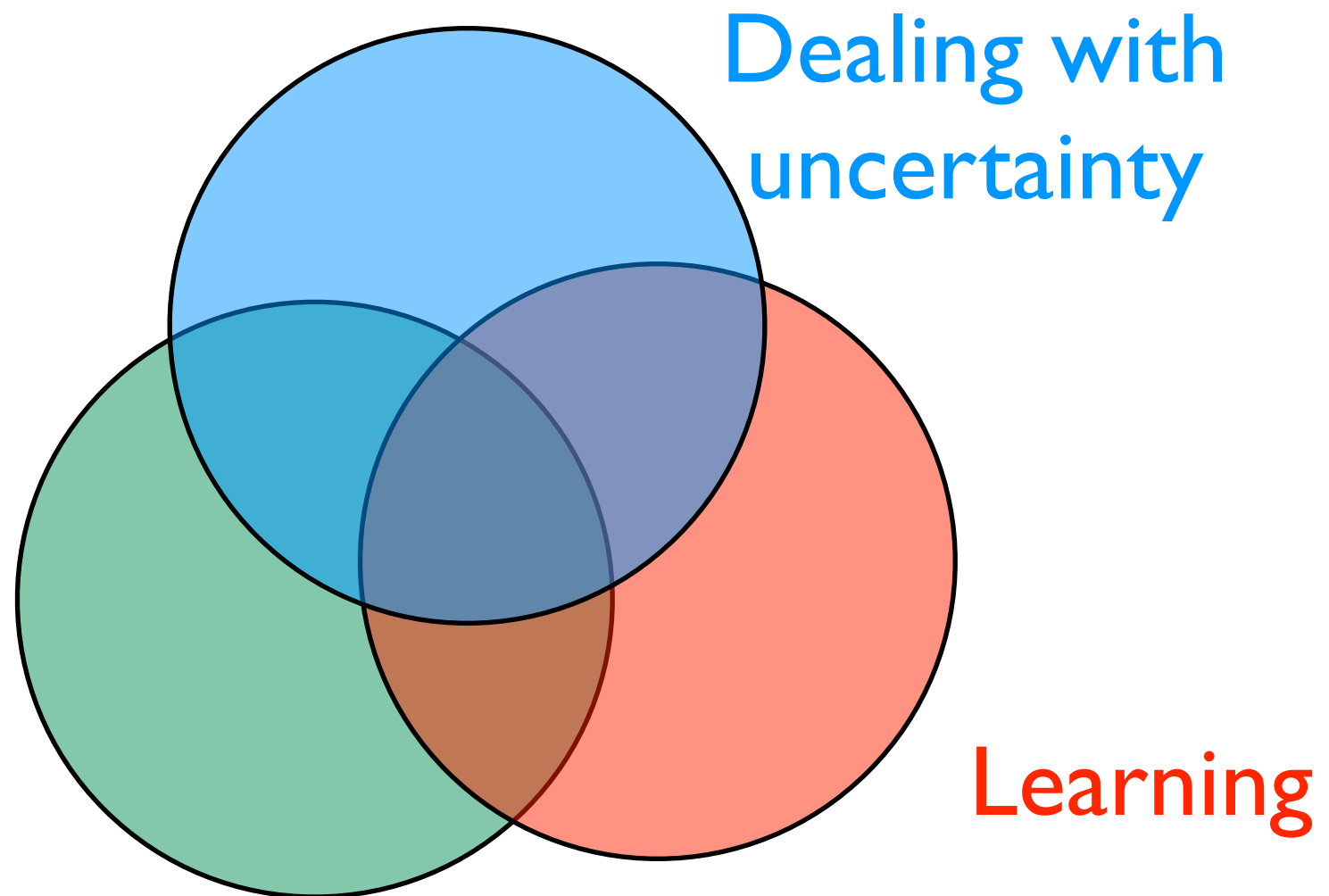
ProbLog

probabilistic Prolog

Prolog / logic
programming

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) , smokes(Y) .
```



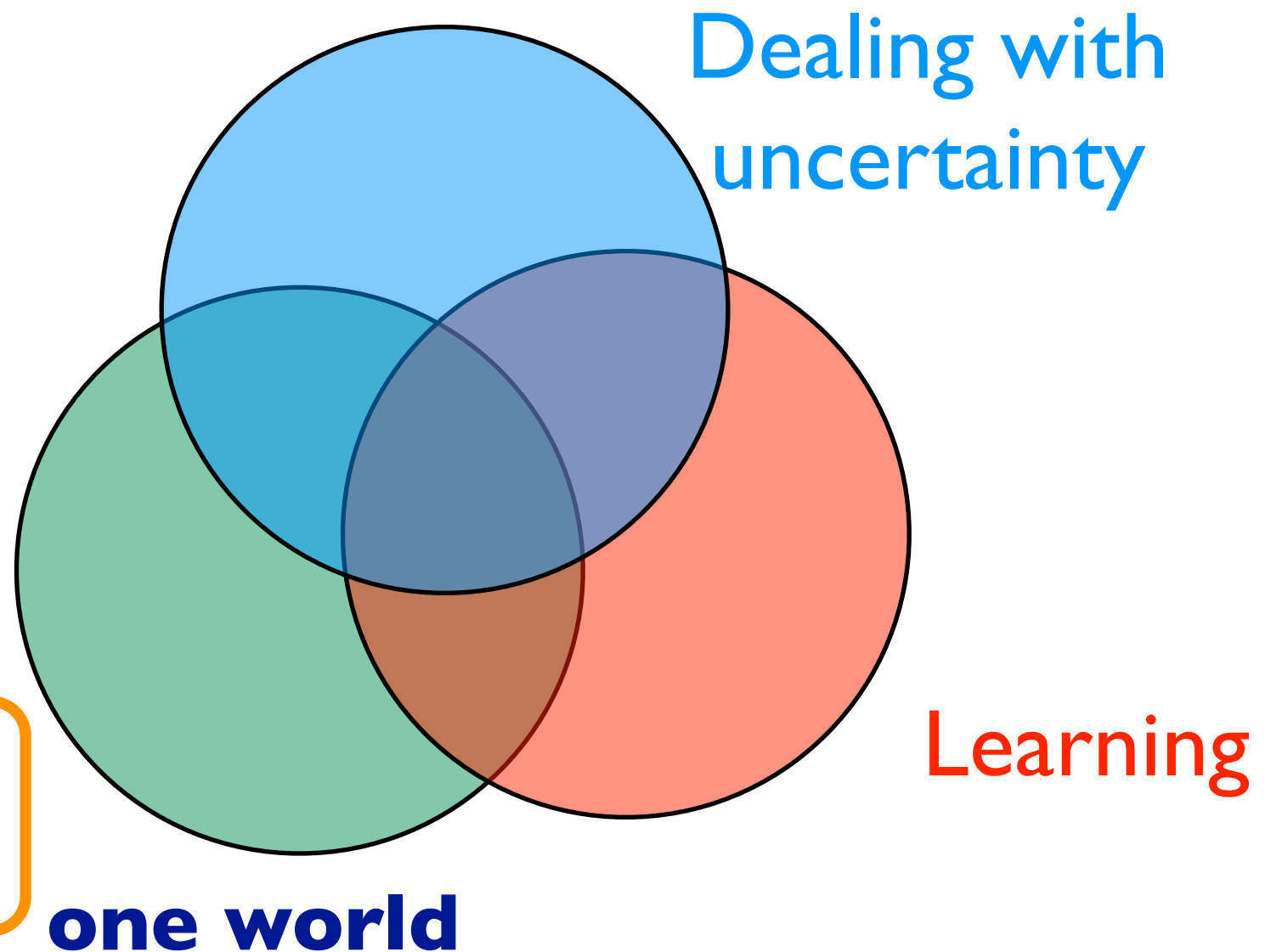
ProbLog

probabilistic Prolog

Prolog / logic
programming

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) , smokes(Y) .
```



ProbLog

probabilistic Prolog

```
0.8::stress(ann).  
0.6::influences(ann,bob).  
0.2::influences(bob,carl).
```

atoms as random
variables

Prolog / logic
programming

```
stress(ann).  
influences(ann,bob).  
influences(bob,carl).
```

one world

Learning

```
smokes(X) :- stress(X).  
smokes(X) :-  
    influences(Y,X), smokes(Y).
```

ProbLog

probabilistic Prolog

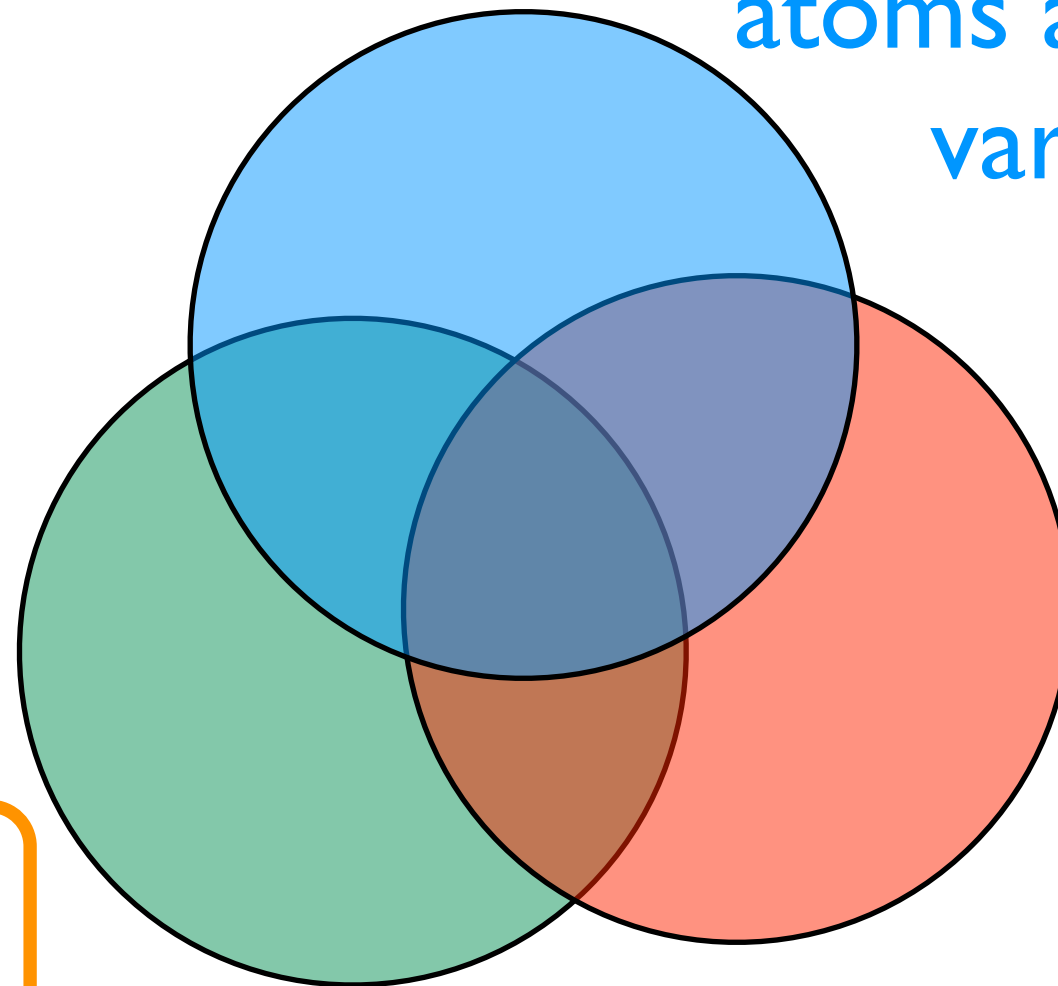
several possible worlds

```
0.8::stress(ann).  
0.6::influences(ann,bob).  
0.2::influences(bob,carl).
```

atoms as random
variables

Prolog / logic
programming

```
stress(ann).  
influences(ann,bob).  
influences(bob,carl).
```



Learning

one world

```
smokes(X) :- stress(X).  
smokes(X) :-  
    influences(Y,X), smokes(Y).
```

ProbLog

probabilistic Prolog

several possible worlds

```
0.8::stress(ann).  
0.6::influences(ann,bob).  
0.2::influences(bob,carl).
```

atoms as random
variables

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds

Prolog / logic
programming

```
stress(ann).  
influences(ann,bob).  
influences(bob,carl).
```

```
smokes(X) :- stress(X).  
smokes(X) :-  
    influences(Y,X), smokes(Y).
```

one world

Learning

ProbLog

probabilistic Prolog

several possible worlds

```
0.8::stress(ann).  
0.6::influences(ann,bob).  
0.2::influences(bob,carl).
```

atoms as random
variables

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds

Prolog / logic
programming

```
stress(ann).  
influences(ann,bob).  
influences(bob,carl).
```

```
smokes(X) :- stress(X).  
smokes(X) :-  
    influences(Y,X), smokes(Y).
```

one world

parameter learning,
adapted relational
learning techniques

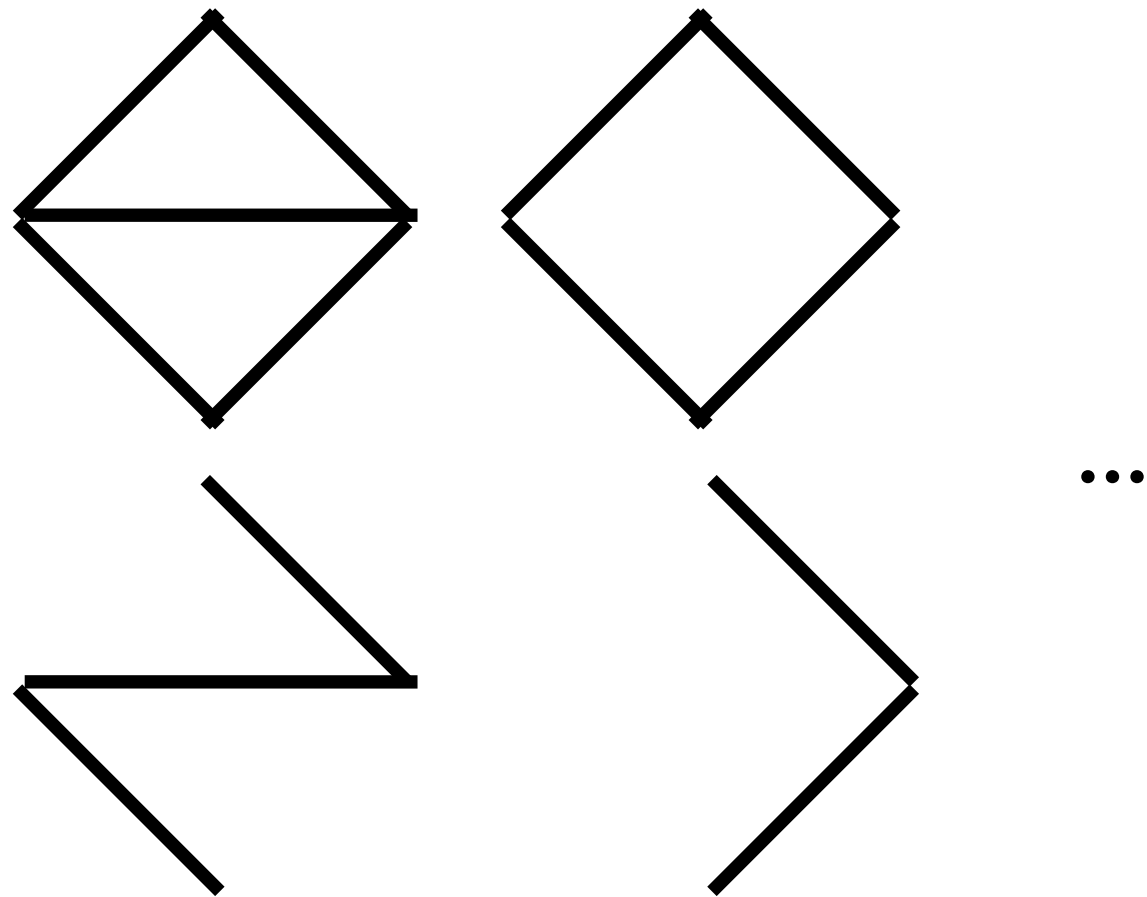
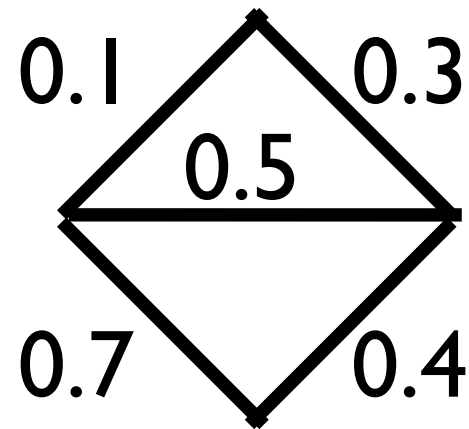
Probabilistic Prologs: Two Views

- Distribution semantics:
 - probability distribution over interpretations
 - degree of belief
- Stochastic Logic Programs (SLPs):
 - probability distribution over query answers
 - like in probabilistic grammars

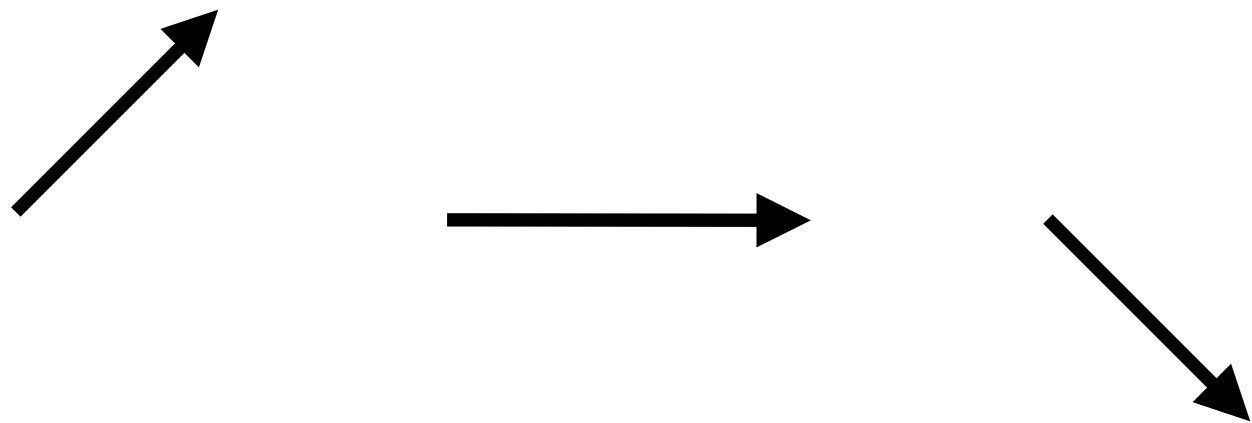
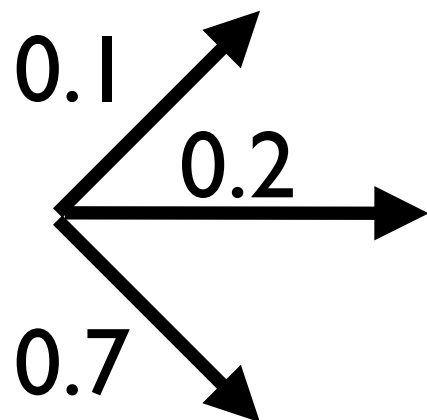
Probabilistic Prologs: Two Views

- Distribution semantics: **random graph**
 - probability distribution over interpretations
 - degree of belief
- Stochastic Logic Programs (SLPs): **random walk**
 - probability distribution over query answers
 - like in probabilistic grammars

Random graph:
which edges exist?



Random walk:
which edge to take
from a given node?



Probabilistic Logic Programming

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds

e.g., PRISM, ICL, ProbLog, LPADs, CP-logic, ...

multi-valued
switches

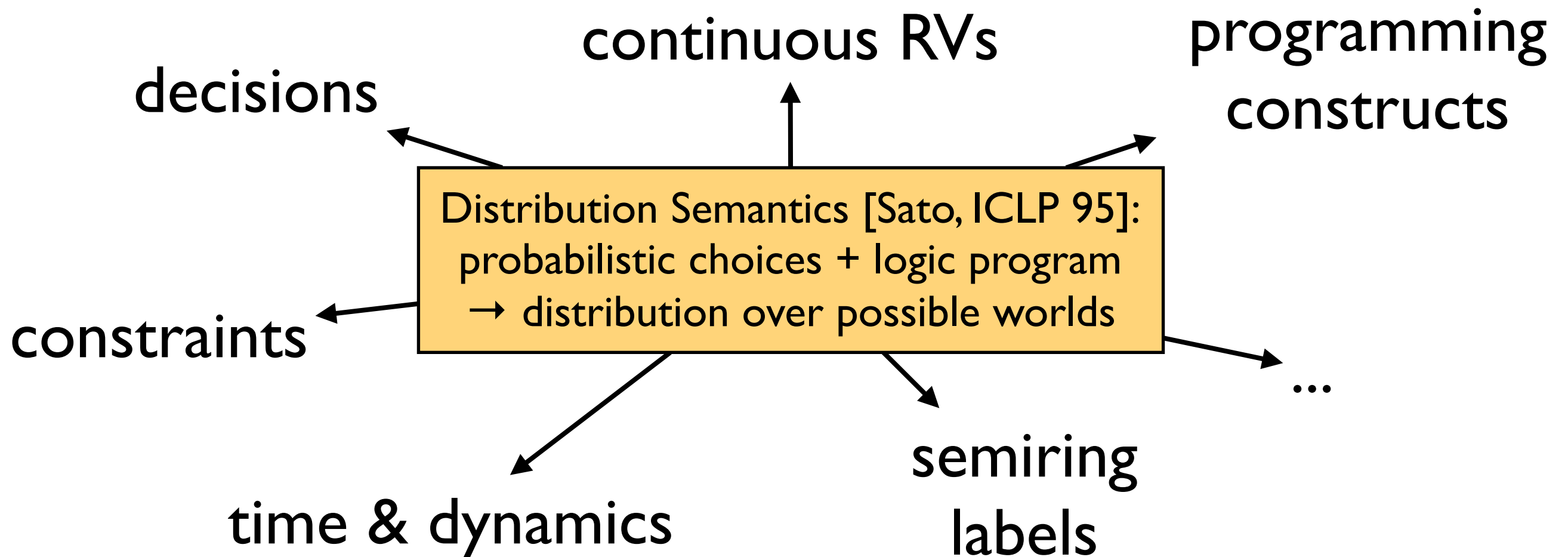
probabilistic
facts

probabilistic
alternatives

annotated
disjunctions

causal-
probabilistic
laws

Extensions of basic PLP



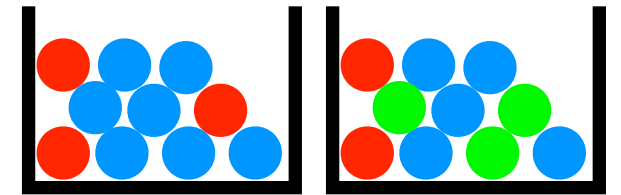
Roadmap

- Modeling
- Reasoning
- Language extensions
- Advanced topics

... with some detours on the way

ProbLog by example:

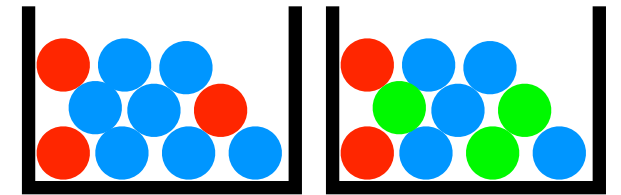
A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

ProbLog by example:

A bit of gambling

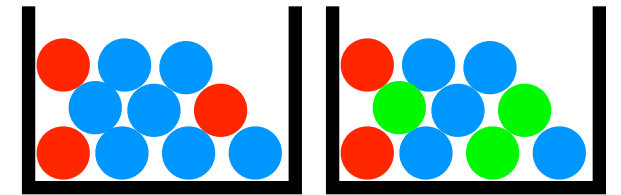


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

0.4 :: heads.

probabilistic fact: heads is true with probability 0.4 (and false with 0.6)

ProbLog by example:

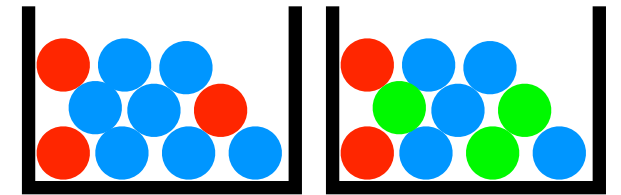


A bit of gambling

- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

`0.4 :: heads.` **annotated disjunction:** first ball is red
with probability 0.3 and blue with 0.7
`0.3 :: col(1,red) ; 0.7 :: col(1,blue) <- true.`

ProbLog by example:



A bit of gambling

- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

0.4 :: heads.

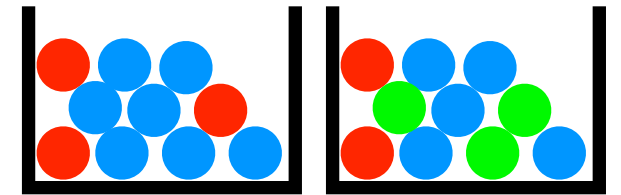
0.3 :: col(1,red) ; 0.7 :: col(1,blue) <- true.

0.2 :: col(2,red) ; 0.3 :: col(2,green) ;
0.5 :: col(2,blue) <- true.

annotated disjunction: second ball is red with probability 0.2, green with 0.3, and blue with 0.5

ProbLog by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.
```

```
0.3 :: col(1,red) ; 0.7 :: col(1,blue) <- true.
```

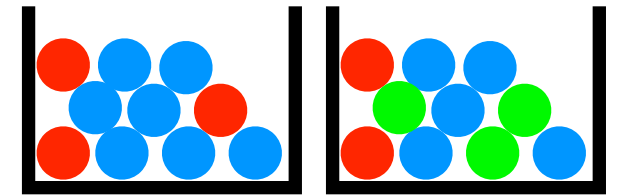
```
0.2 :: col(2,red) ; 0.3 :: col(2,green) ;  
0.5 :: col(2,blue) <- true.
```

```
win :- heads, col(_,red).
```

logical rule encoding
background knowledge

ProbLog by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.
```

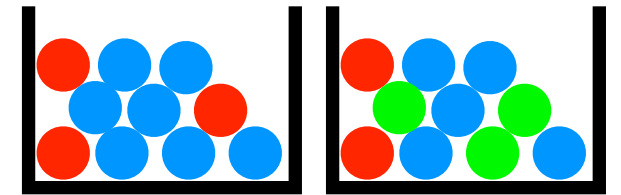
```
0.3 :: col(1,red) ; 0.7 :: col(1,blue) <- true.
```

```
0.2 :: col(2,red) ; 0.3 :: col(2,green) ;  
0.5 :: col(2,blue) <- true.
```

```
win :- heads, col(_,red).  
win :- col(1,C), col(2,C).  
logical rule encoding background knowledge
```

ProbLog by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.
```

probabilistic choices

```
0.3 :: col(1,red) ; 0.7 :: col(1,blue) <- true.
```

```
0.2 :: col(2,red) ; 0.3 :: col(2,green) ;  
0.5 :: col(2,blue) <- true.
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

consequences

Questions

```
0.4 :: heads.
```

```
0.3 :: col(1,red) ; 0.7 :: col(1,blue) <- true.
```

```
0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue) <- true.
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

- Probability of **win**?
- Probability of **win** given **col(2,green)**?
- Most probable world where **win** is true?

Questions

```
0.4 :: heads.
```

```
0.3 :: col(1,red) ; 0.7 :: col(1,blue) <- true.
```

```
0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue) <- true.
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

marginal probability

- Probability of **win**
query
- Probability of **win** given **col(2,green)**?
- Most probable world where **win** is true?

Questions

```
0.4 :: heads.
```

```
0.3 :: col(1,red) ; 0.7 :: col(1,blue) <- true.
```

```
0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue) <- true.
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

marginal probability

- Probability of `win`?

conditional probability

- Probability of `win` given `col(2,green)`?

evidence

- Most probable world where `win` is true?

Questions

```
0.4 :: heads.
```

```
0.3 :: col(1,red) ; 0.7 :: col(1,blue) <- true.
```

```
0.2 :: col(2,red) ; 0.3 :: col(2,green) ; 0.5 :: col(2,blue) <- true.
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

marginal probability

- Probability of `win`?

conditional probability

- Probability of `win` given `col(2,green)`?

- Most probable world where `win` is true?

MPE inference

Possible Worlds

0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.

0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).

win :- col(1,C), col(2,C).

Possible Worlds

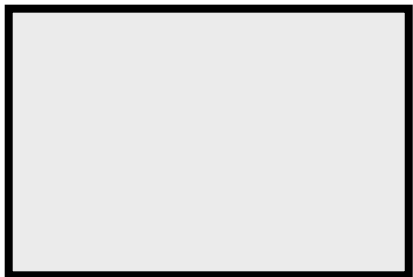
```
0.4 :: heads.
```

```
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```



Possible Worlds

`0.4 :: heads.`

`0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.`

`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.`

`win :- heads, col(_,red).`

`win :- col(1,C), col(2,C).`

0.4



Possible Worlds

```
0.4 :: heads.
```

```
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

0.4×0.3



Possible Worlds

```
0.4 :: heads.
```

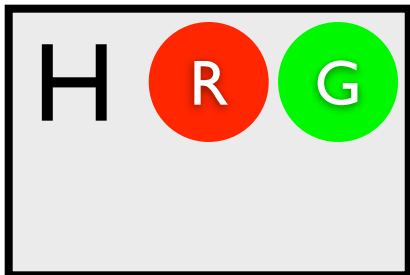
```
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.
```

```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



Possible Worlds

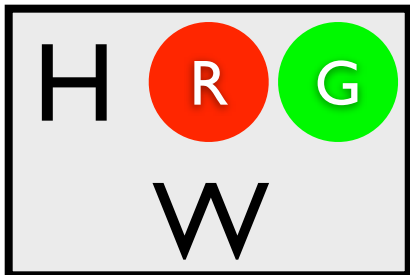
`0.4 :: heads.`

`0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.`

`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.`

`win :- heads, col(_,red).`
`win :- col(1,C), col(2,C).`

$0.4 \times 0.3 \times 0.3$



Possible Worlds

`0.4 :: heads.`

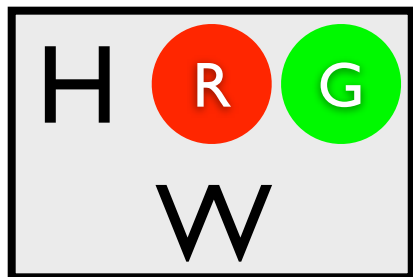
`0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.`

`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.`

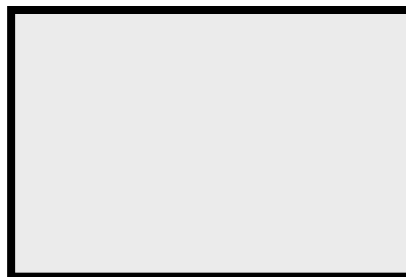
`win :- heads, col(_,red).`

`win :- col(1,C), col(2,C).`

$0.4 \times 0.3 \times 0.3$



$(1 - 0.4)$



Possible Worlds

```
0.4 :: heads.
```

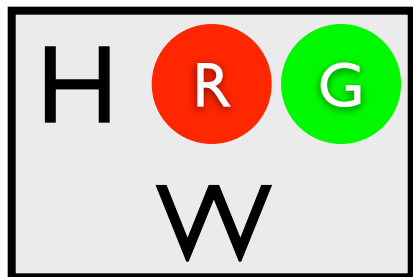
```
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.
```

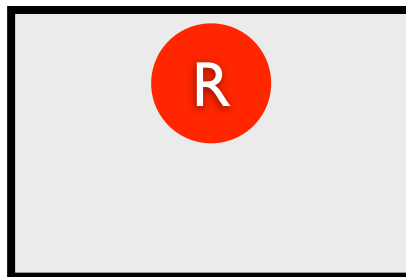
```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3$



Possible Worlds

```
0.4 :: heads.
```

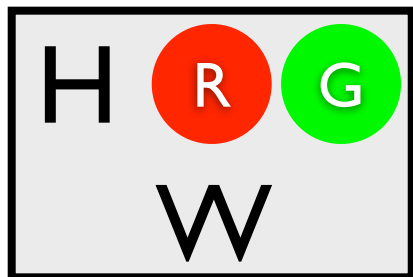
```
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.
```

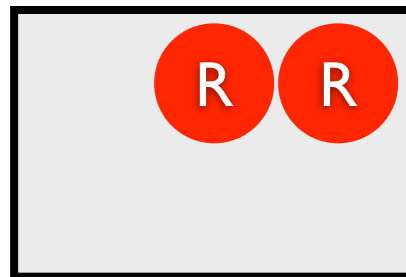
```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

$$0.4 \times 0.3 \times 0.3$$



$$(1-0.4) \times 0.3 \times 0.2$$



Possible Worlds

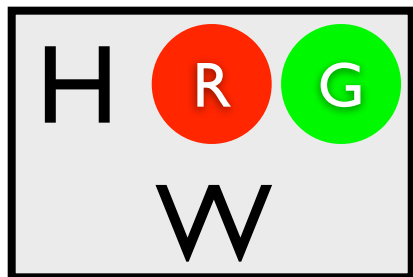
`0.4 :: heads.`

`0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.`

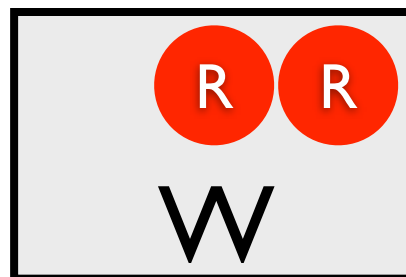
`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.`

`win :- heads, col(_,red).`
`win :- col(1,C), col(2,C).`

$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



Possible Worlds

`0.4 :: heads.`

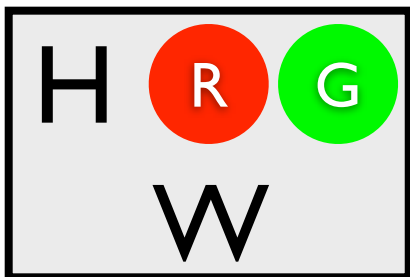
`0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.`

`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.`

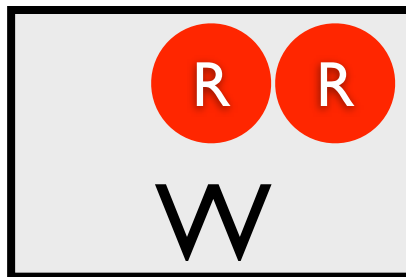
`win :- heads, col(_,red).`

`win :- col(1,C), col(2,C).`

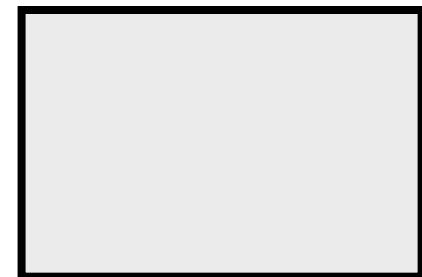
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



$(1-0.4)$



Possible Worlds

`0.4 :: heads.`

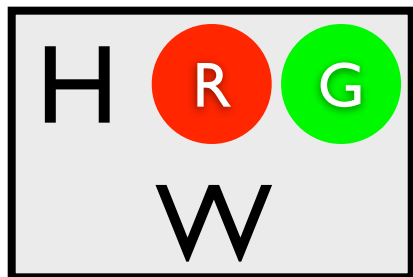
`0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.`

`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.`

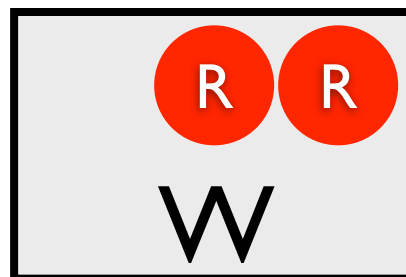
`win :- heads, col(_,red).`

`win :- col(1,C), col(2,C).`

$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



$(1-0.4) \times 0.3$



Possible Worlds

```
0.4 :: heads.
```

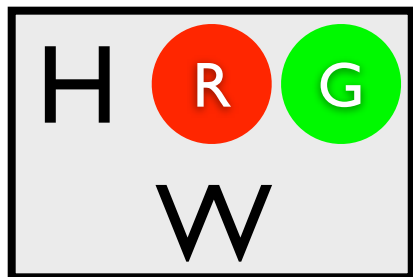
```
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.
```

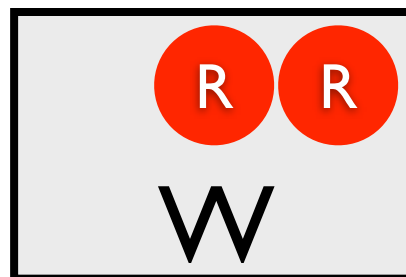
```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

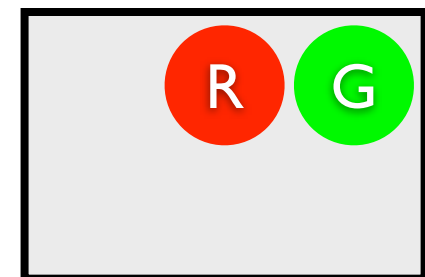
$$0.4 \times 0.3 \times 0.3$$



$$(1-0.4) \times 0.3 \times 0.2$$



$$(1-0.4) \times 0.3 \times 0.3$$



Possible Worlds

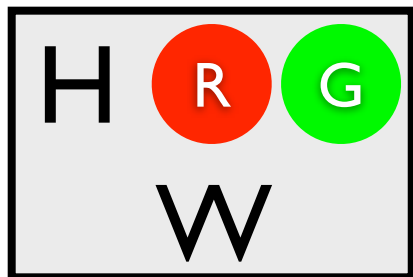
`0.4 :: heads.`

`0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.`

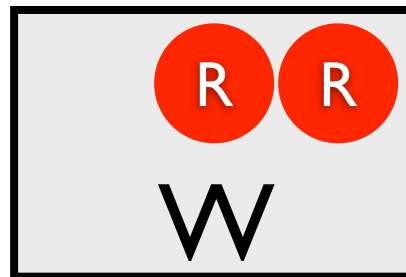
`0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.`

```
win :- heads, col(_,red).  
win :- col(1,C), col(2,C).
```

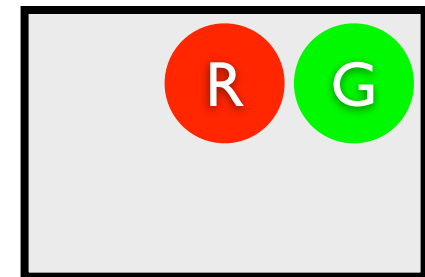
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



$(1-0.4) \times 0.3 \times 0.3$



Possible Worlds

```
0.4 :: heads.
```

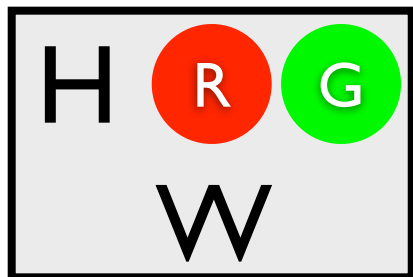
```
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
```

```
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.
```

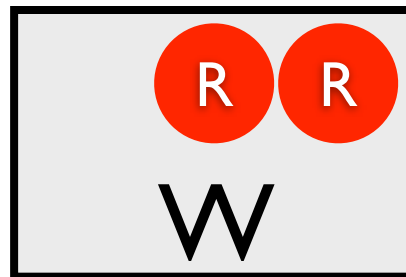
```
win :- heads, col(_,red).
```

```
win :- col(1,C), col(2,C).
```

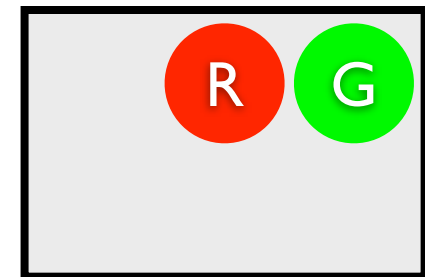
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$

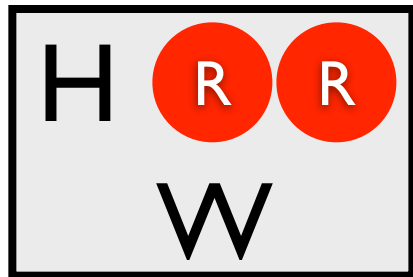


$(1-0.4) \times 0.3 \times 0.3$

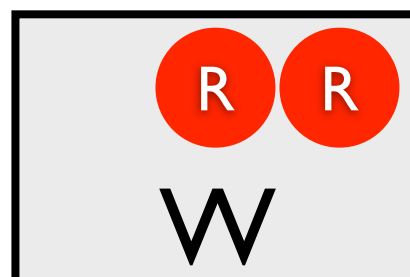


All Possible Worlds

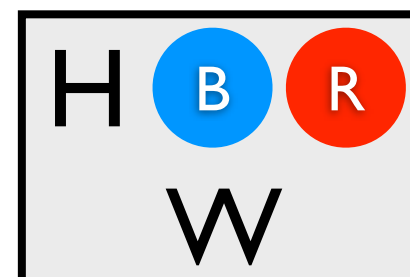
0.024



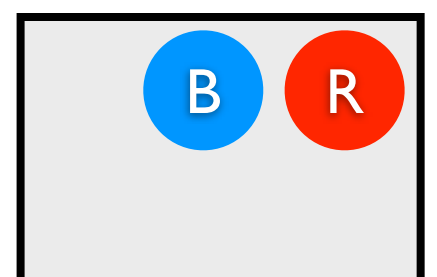
0.036



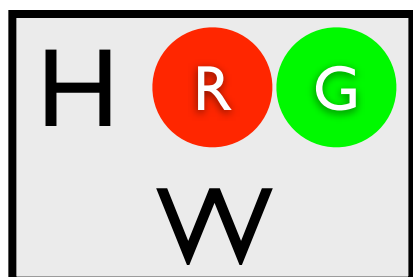
0.056



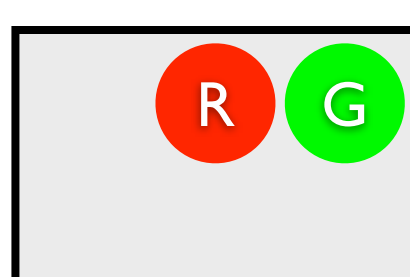
0.084



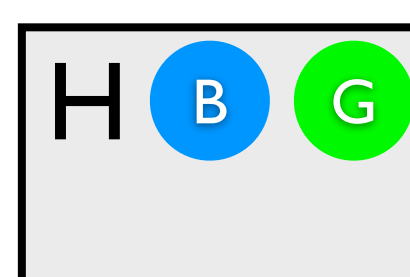
0.036



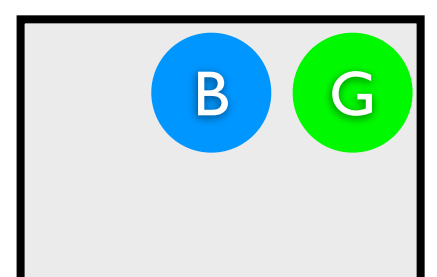
0.054



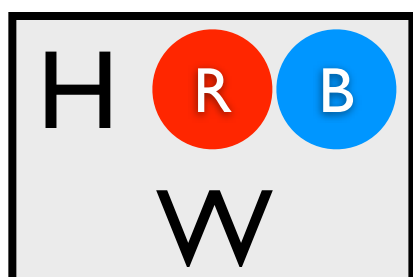
0.084



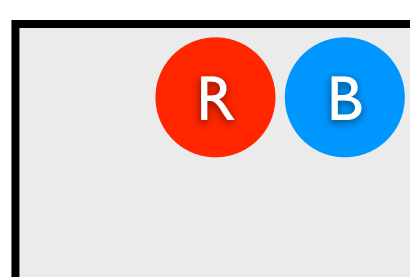
0.126



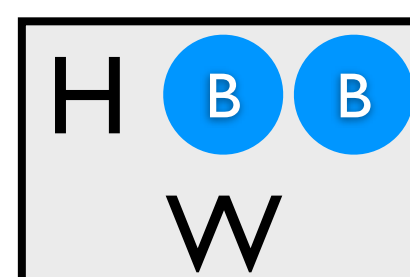
0.060



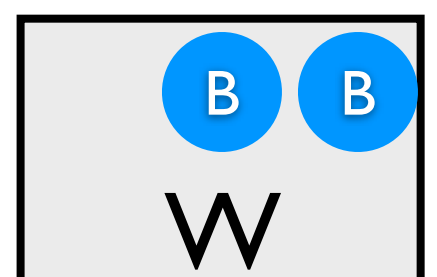
0.090



0.140



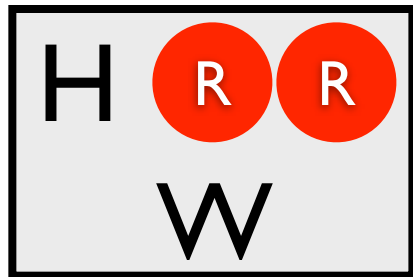
0.210



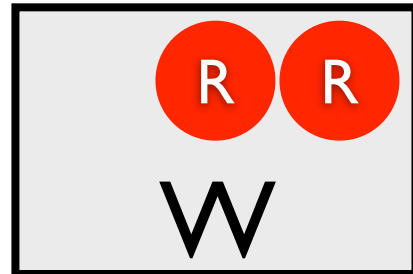
Most likely world where `win` is true?

MPE Inference

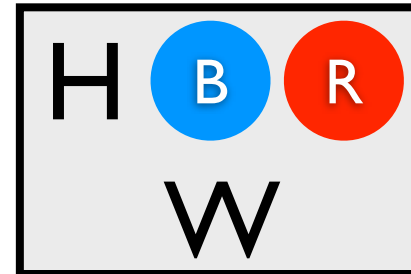
0.024



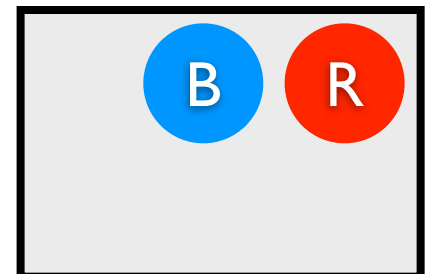
0.036



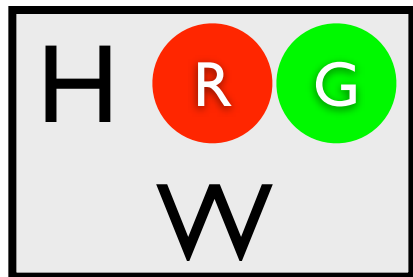
0.056



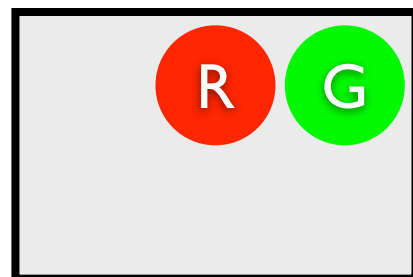
0.084



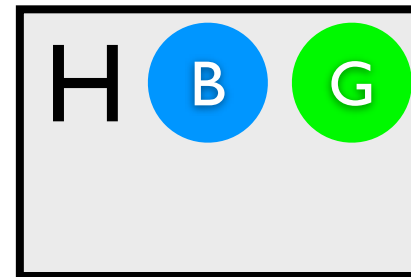
0.036



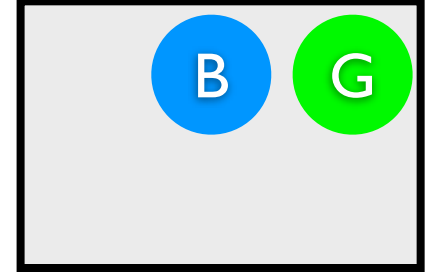
0.054



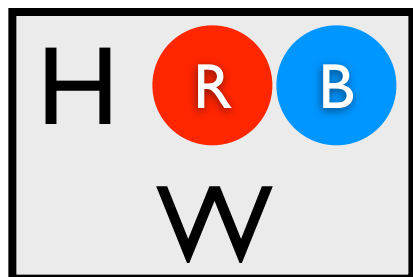
0.084



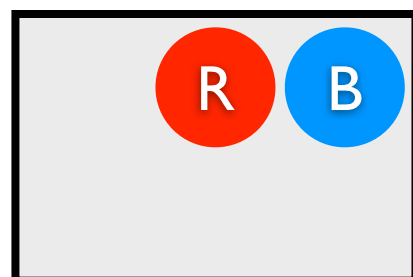
0.126



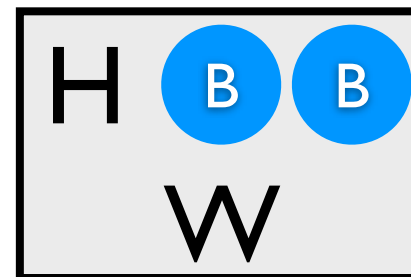
0.060



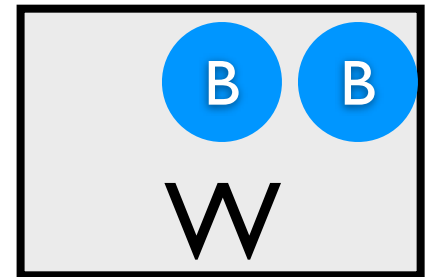
0.090



0.140



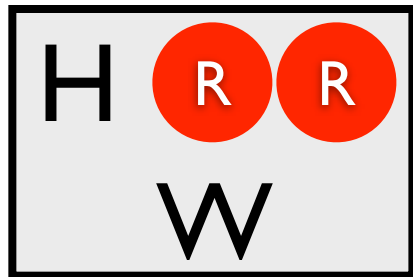
0.210



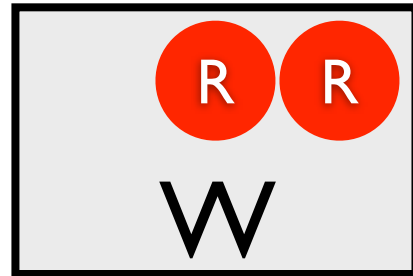
Most likely world where `win` is true?

MPE Inference

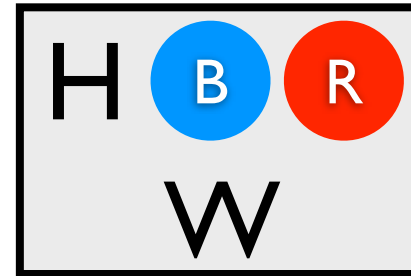
0.024



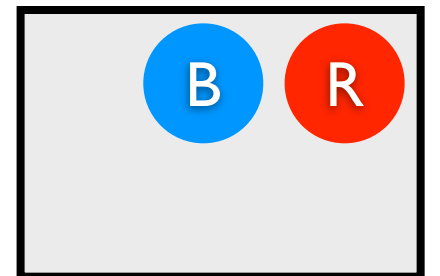
0.036



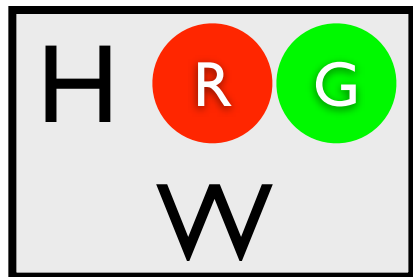
0.056



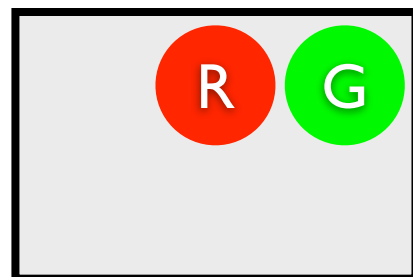
0.084



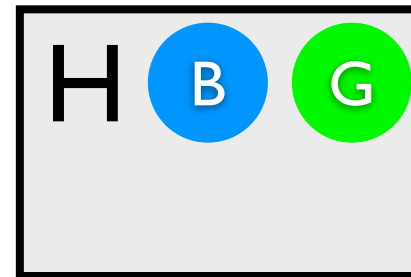
0.036



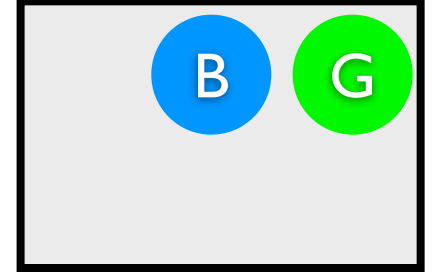
0.054



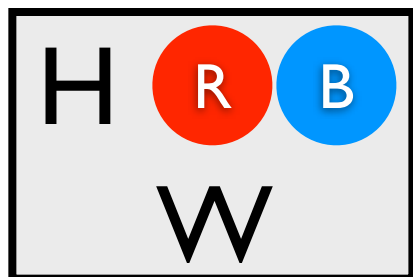
0.084



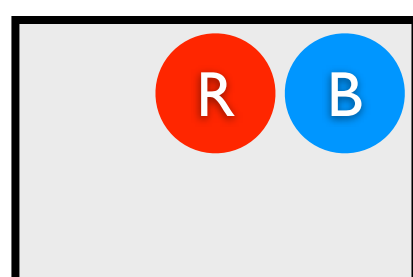
0.126



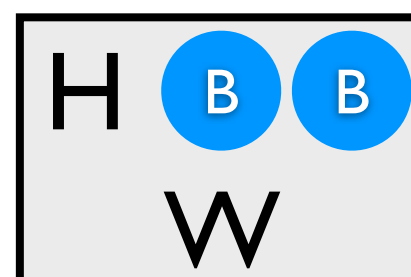
0.060



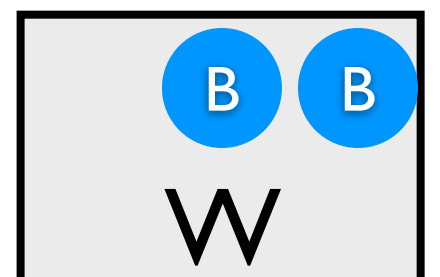
0.090



0.140



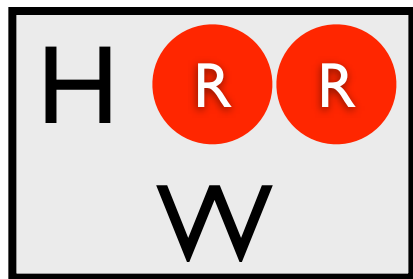
0.210



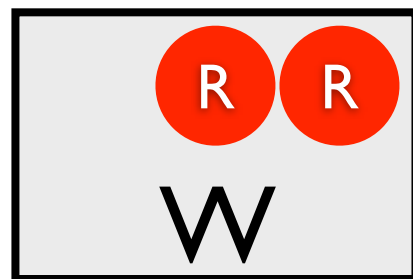
Most likely world where `col(2, blue)` is false?

MPE Inference

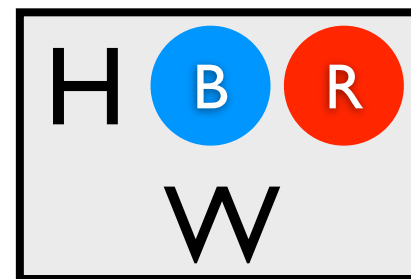
0.024



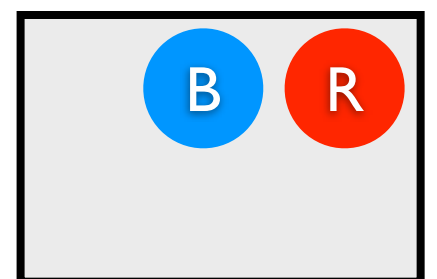
0.036



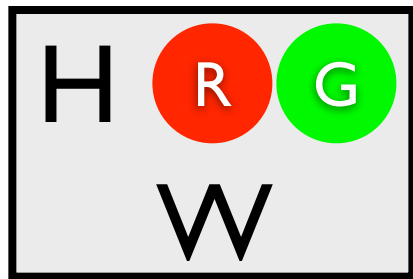
0.056



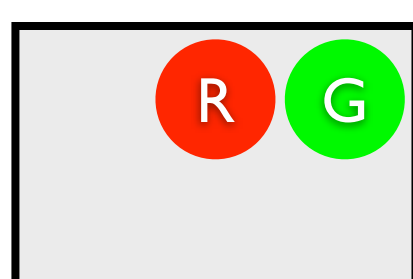
0.084



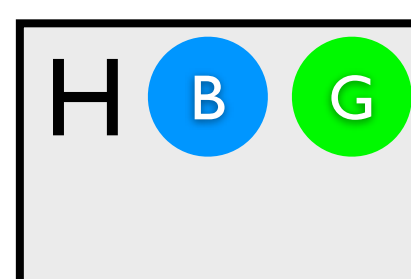
0.036



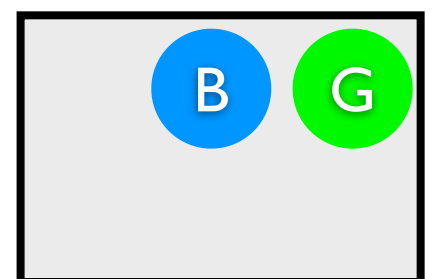
0.054



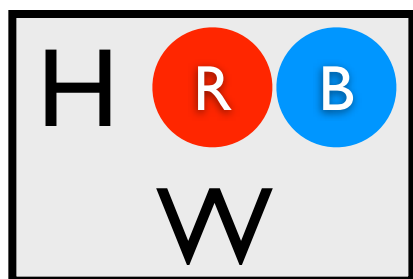
0.084



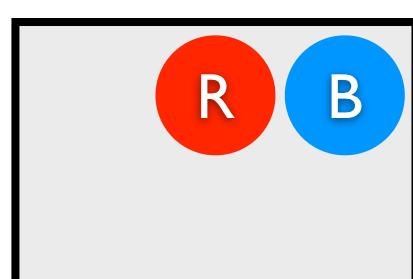
0.126



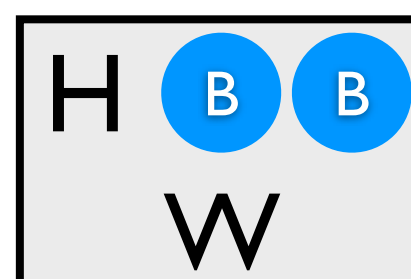
0.060



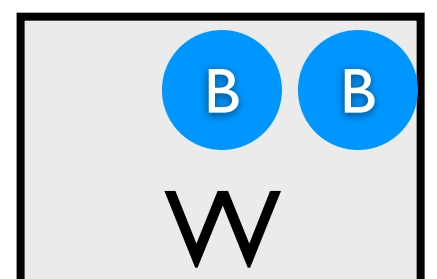
0.090



0.140



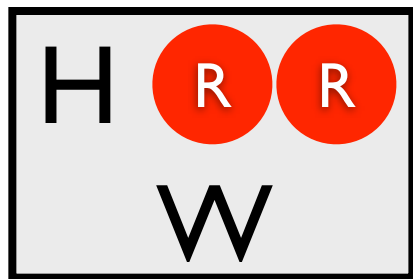
0.210



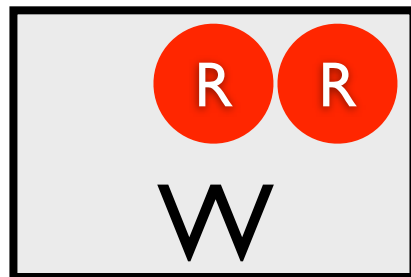
Most likely world where `col(2, blue)` is false?

MPE Inference

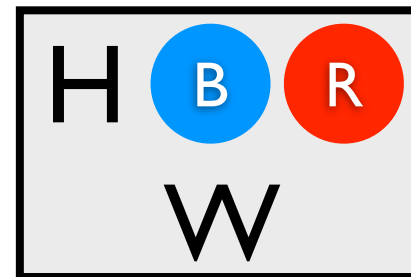
0.024



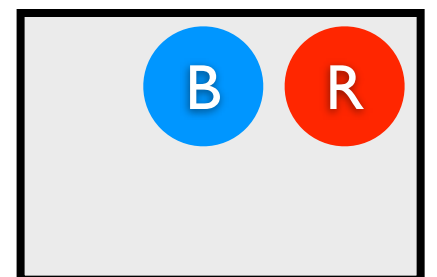
0.036



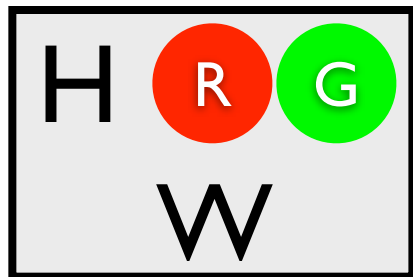
0.056



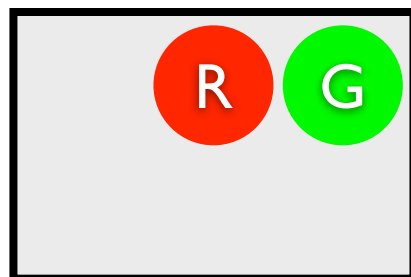
0.084



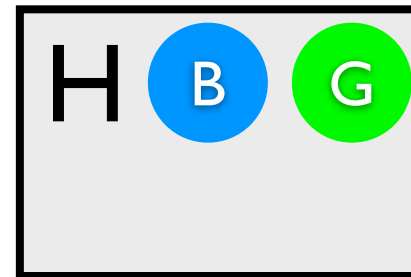
0.036



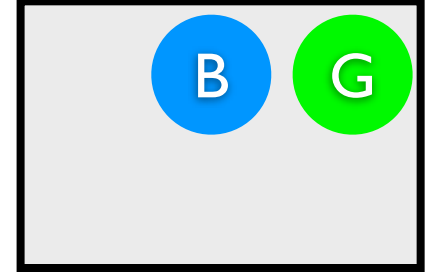
0.054



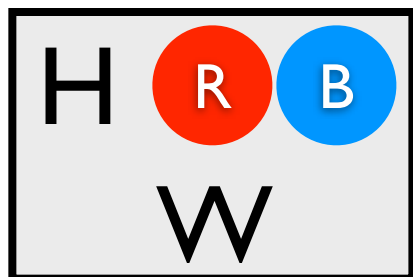
0.084



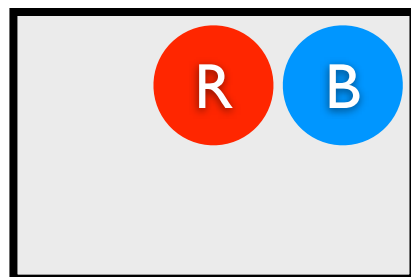
0.126



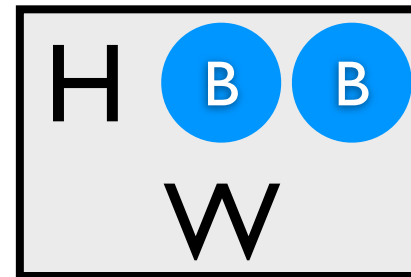
0.060



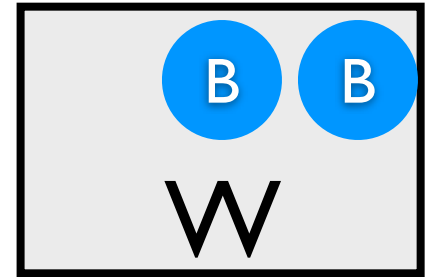
0.090



0.140



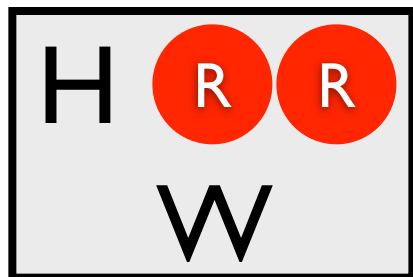
0.210



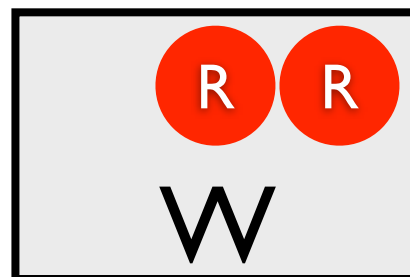
$$P(\text{win}) = ?$$

Marginal
Probability

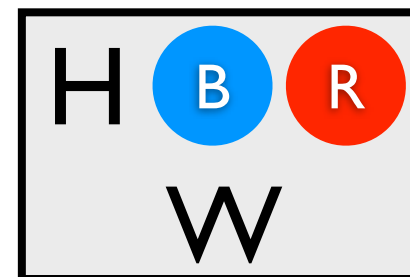
0.024



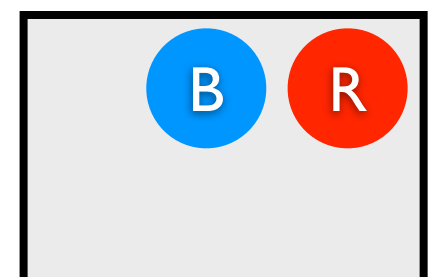
0.036



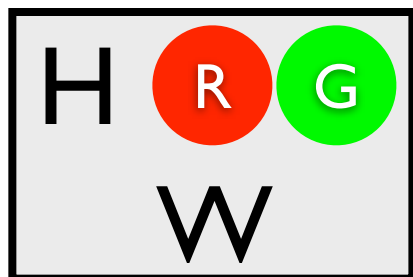
0.056



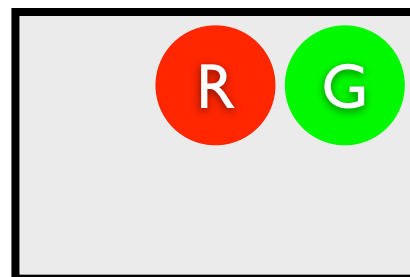
0.084



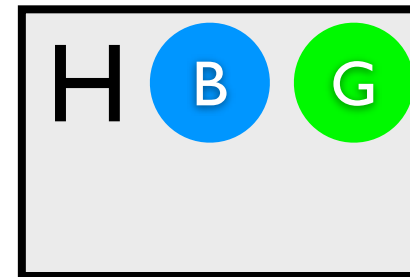
0.036



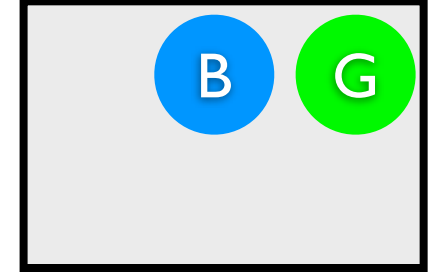
0.054



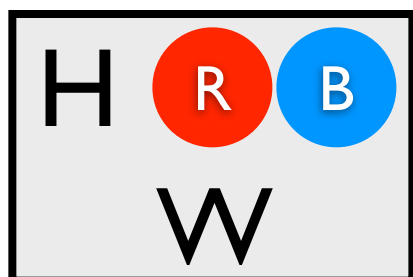
0.084



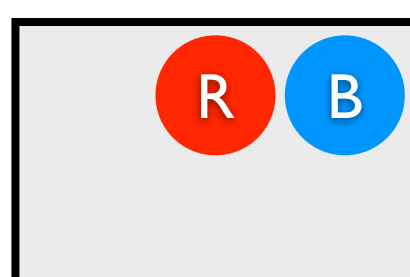
0.126



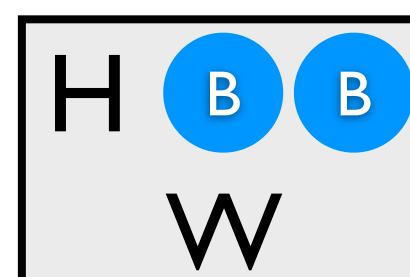
0.060



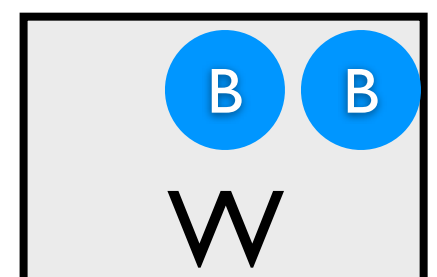
0.090



0.140



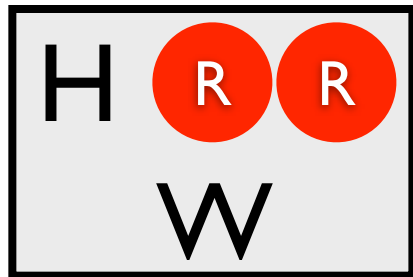
0.210



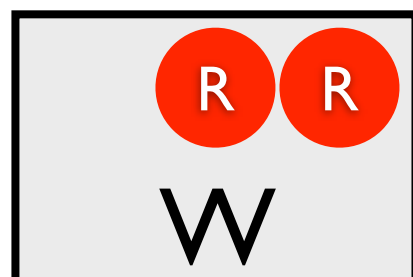
$$P(\text{win}) = \Sigma$$

Marginal
Probability

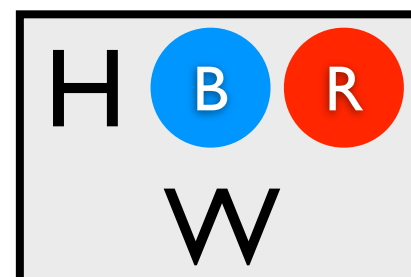
0.024



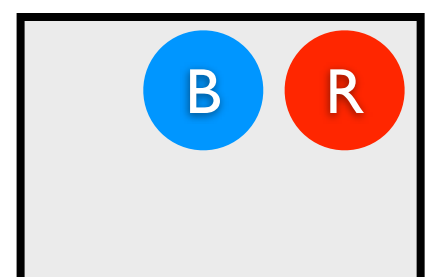
0.036



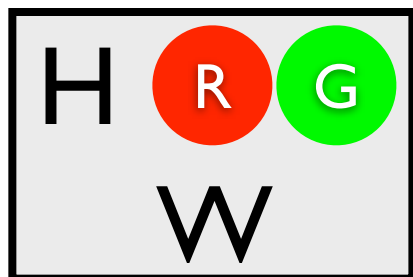
0.056



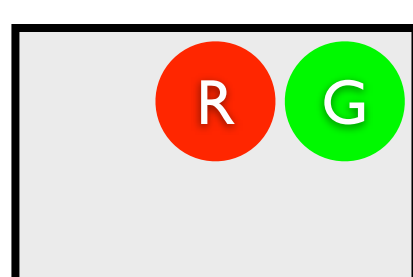
0.084



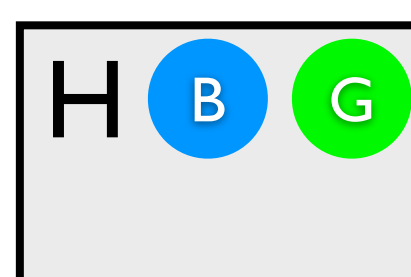
0.036



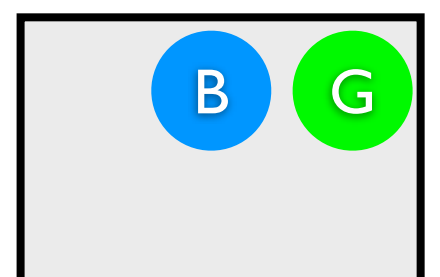
0.054



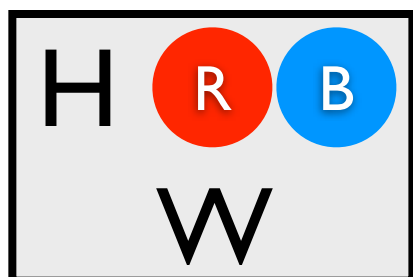
0.084



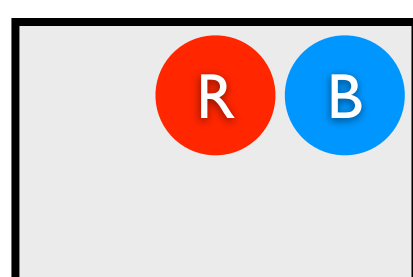
0.126



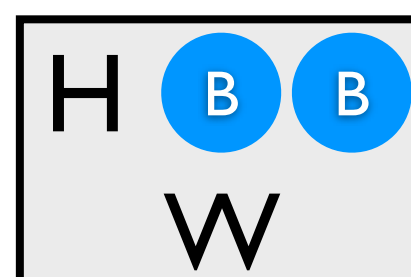
0.060



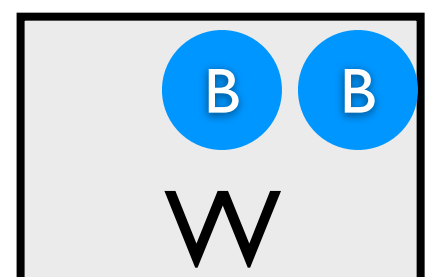
0.090



0.140



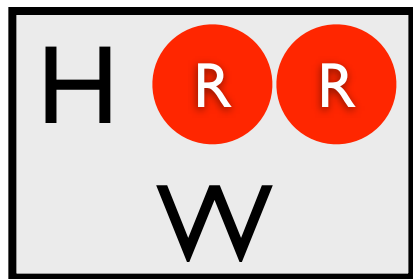
0.210



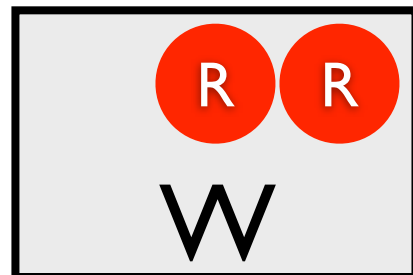
$$P(\text{win}) = \sum = 0.562$$

Marginal
Probability

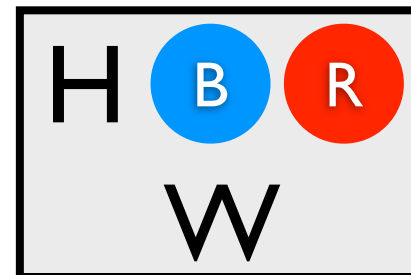
0.024



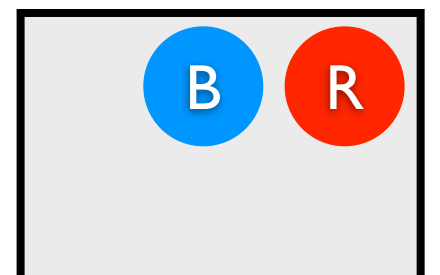
0.036



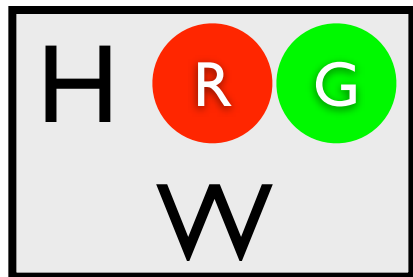
0.056



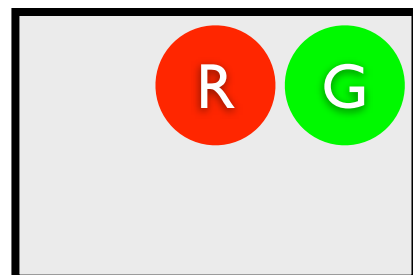
0.084



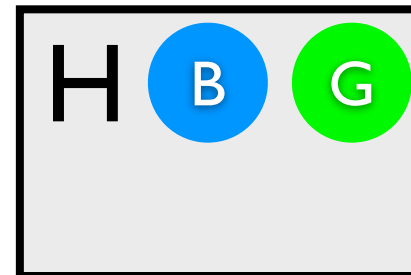
0.036



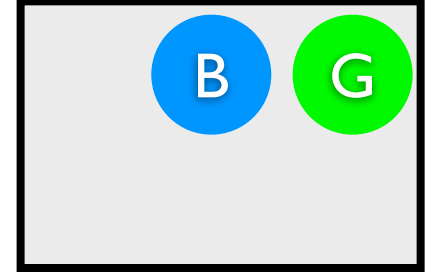
0.054



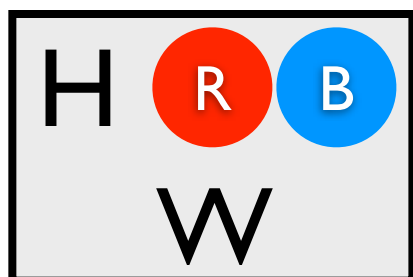
0.084



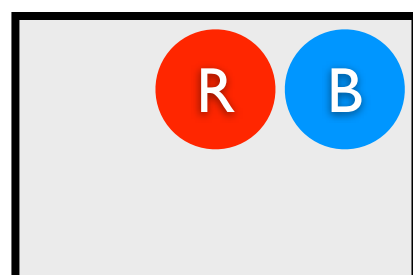
0.126



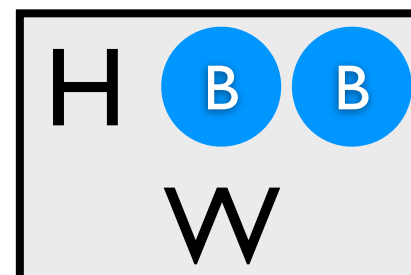
0.060



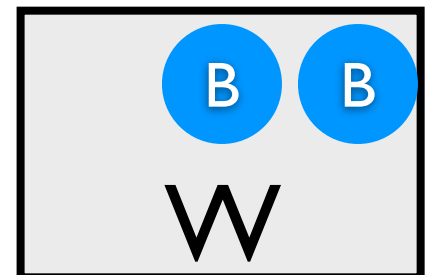
0.090



0.140



0.210



$P(\text{win}|\text{col}(2,\text{green})) = ?$

Conditional Probability

0.024

H

R

R

W

0.036

R

R

W

0.056

H

B

R

W

0.084

B

R

0.036

H

R

G

W

0.054

R

G

0.084

H

B

G

0.126

B

G

0.060

H

R

B

W

0.090

R

B

0.140

H

B

B

W

0.210

B

B

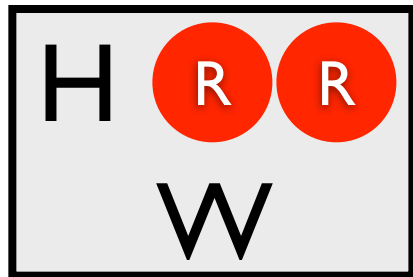
W

$$P(\text{win}|\text{col}(2,\text{green})) = \frac{\sum}{\Sigma}$$

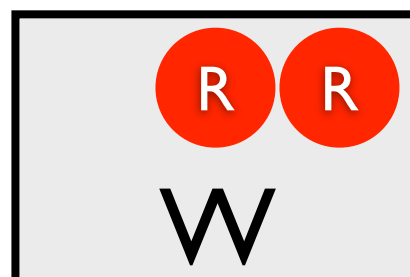
$$= \frac{P(\text{win} \wedge \text{col}(2,\text{green}))}{P(\text{col}(2,\text{green}))}$$

Conditional Probability

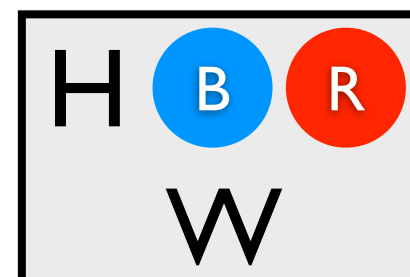
0.024



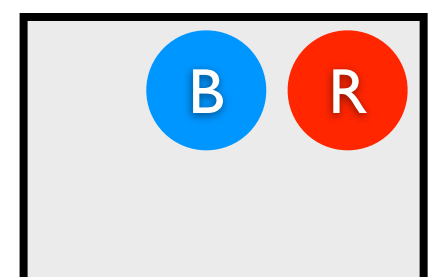
0.036



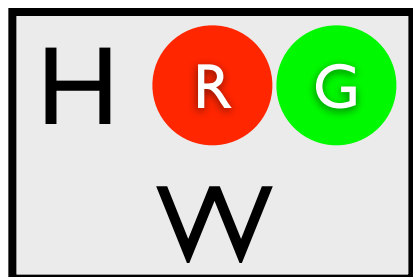
0.056



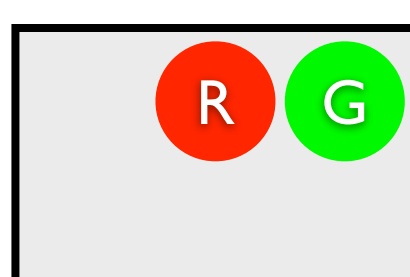
0.084



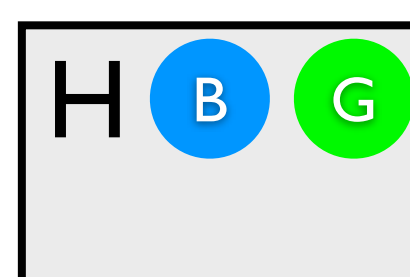
0.036



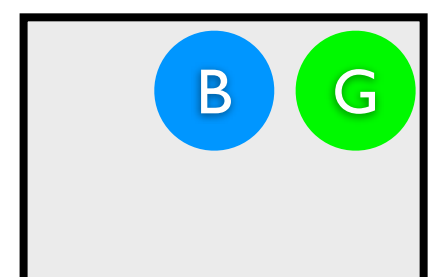
0.054



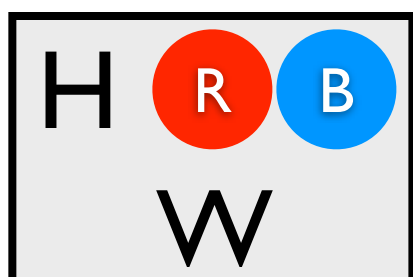
0.084



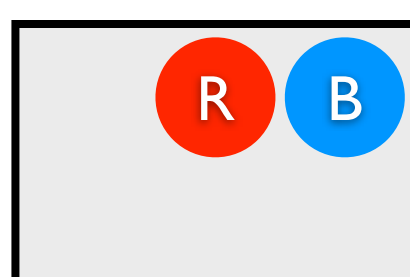
0.126



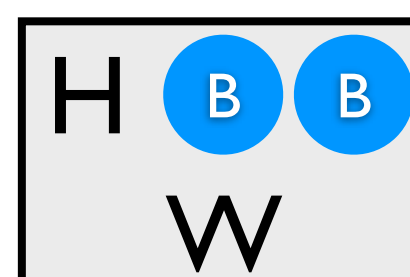
0.060



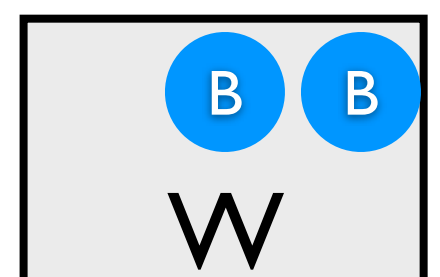
0.090



0.140



0.210

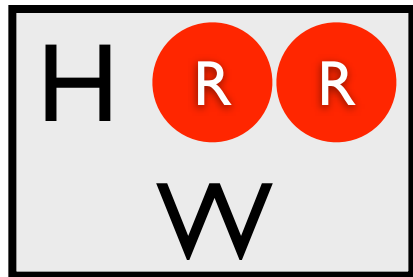


$$P(\text{win}|\text{col}(2,\text{green})) = \frac{\sum}{\Sigma}$$

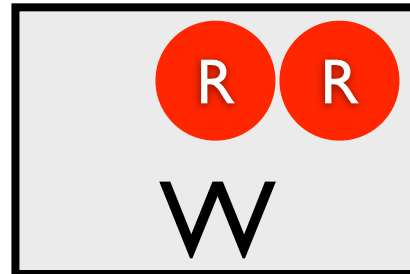
$$= \frac{P(\text{win} \wedge \text{col}(2,\text{green}))}{P(\text{col}(2,\text{green}))}$$

Conditional Probability

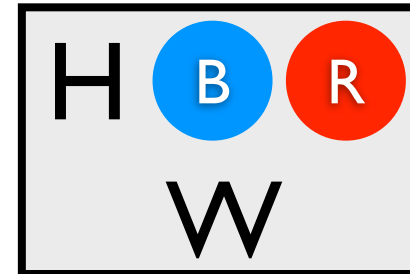
0.024



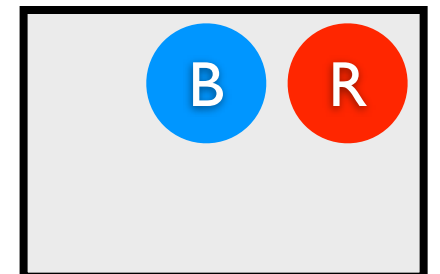
0.036



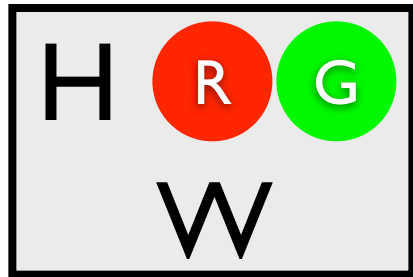
0.056



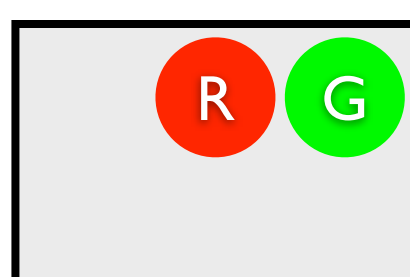
0.084



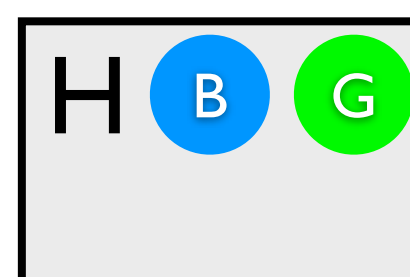
0.036



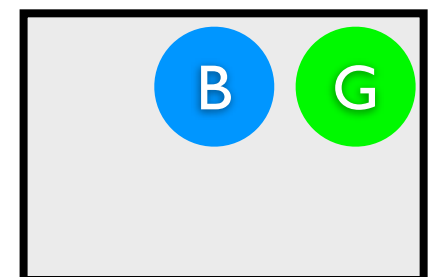
0.054



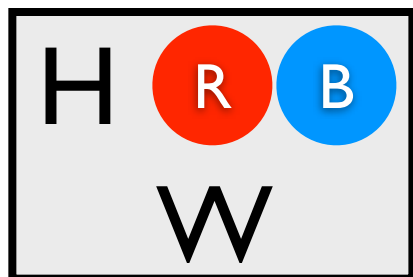
0.084



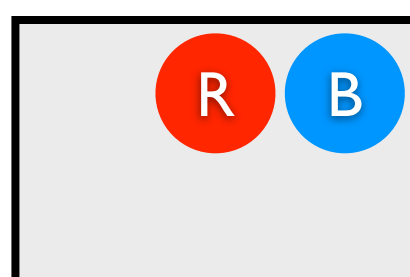
0.126



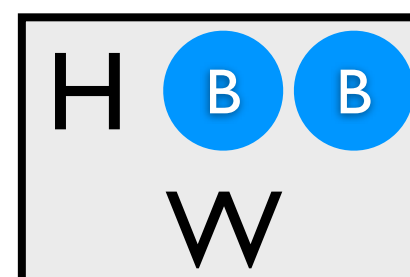
0.060



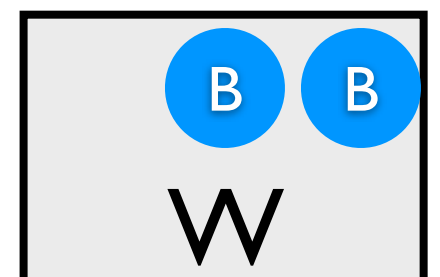
0.090



0.140



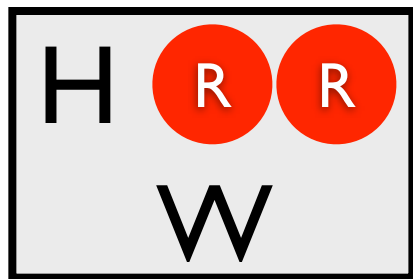
0.210



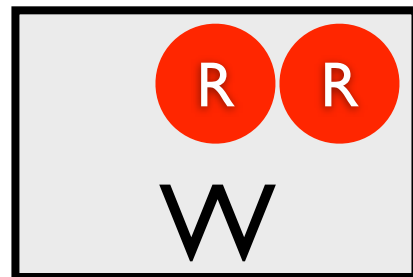
$$P(\text{win}|\text{col}(2,\text{green})) = \frac{\Sigma}{\Sigma} = 0.036/0.3 = 0.12$$

Conditional Probability

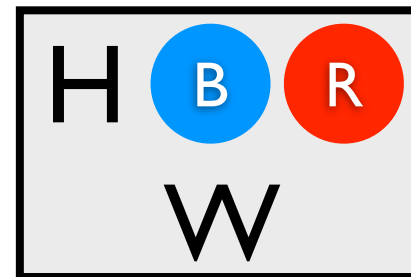
0.024



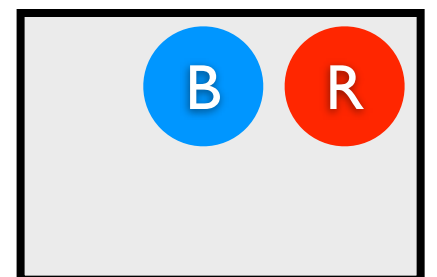
0.036



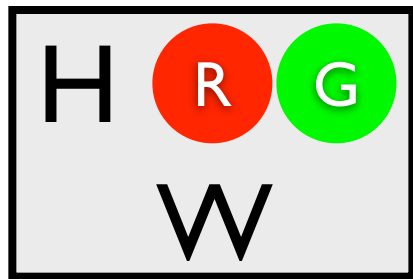
0.056



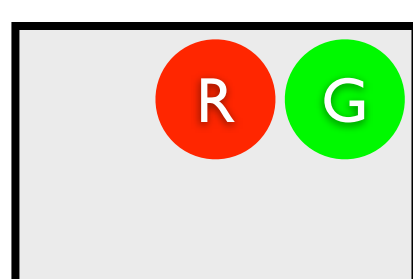
0.084



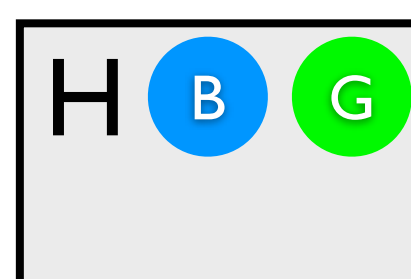
0.036



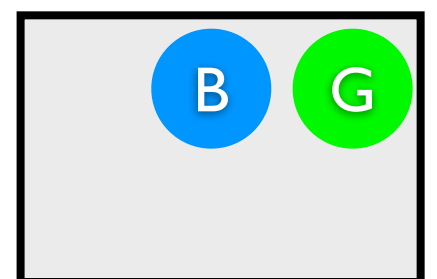
0.054



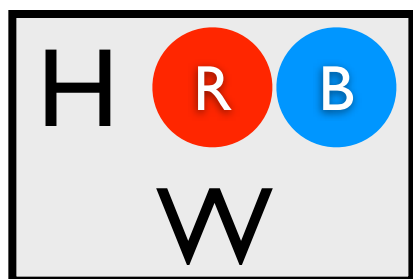
0.084



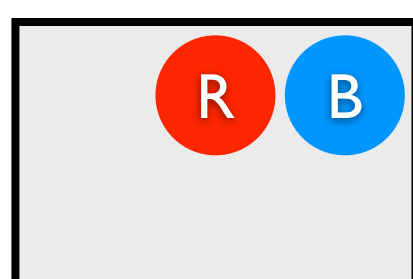
0.126



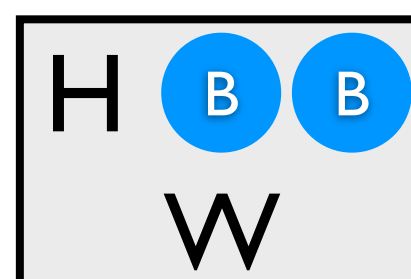
0.060



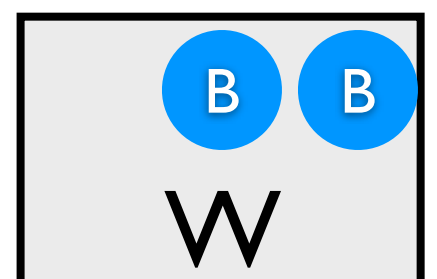
0.090



0.140



0.210



Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]


$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query


$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

subset of
probabilistic
facts

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

subset of probabilistic facts

Prolog rules

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

sum over possible worlds
where Q is true

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

subset of
probabilistic
facts

Prolog
rules

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

sum over possible worlds
where Q is true

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

subset of
probabilistic
facts

Prolog
rules

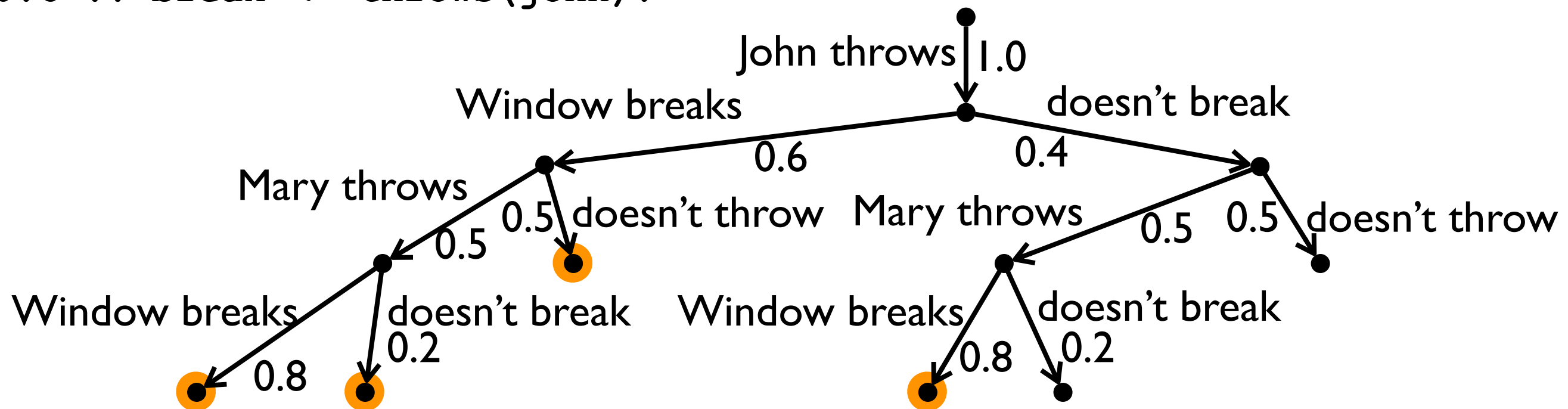
probability of
possible world

Alternative view: CP-Logic

```
throws(john) .
0.5 :: throws(mary) .
```

probabilistic causal laws

```
0.8 :: break <- throws(mary) .
0.6 :: break <- throws(john) .
```



$$P(\text{break}) = 0.6 \times 0.5 \times 0.8 + 0.6 \times 0.5 \times 0.2 + 0.6 \times 0.5 + 0.4 \times 0.5 \times 0.8$$

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

random variable with Gaussian distribution

```
length(Obj) ~ gaussian(6.0, 0.45) :- type(Obj, glass) .
```



Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).  
stackable(OBot,OTop) :-
```

```
     $\simeq \text{length}(\text{OBot}) \geq \simeq \text{length}(\text{OTop}) ,$   
     $\simeq \text{width}(\text{OBot}) \geq \simeq \text{width}(\text{OTop}) .$ 
```

comparing values of
random variables



Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
```

```
stackable(ObjBot,ObjTop) :-
```

```
    ≈length(ObjBot) ≥ ≈length(ObjTop),
```

```
    ≈width(ObjBot) ≥ ≈width(ObjTop).
```

```
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,  
                             0 : pitcher, 0.8676 : plate,  
                             0.0284 : bowl, 0 : serving,  
                             0.1016 : none])
```

```
:- obj(Obj), on(Obj,O2), type(O2,plate).
```

**random variable with
discrete distribution**



Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
    ≈length(OBot) ≥ ≈length(OTop),
    ≈width(OBot) ≥ ≈width(OTop).
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,
                             0 : pitcher, 0.8676 : plate,
                             0.0284 : bowl, 0 : serving,
                             0.1016 : none])
:- obj(Obj), on(Obj,O2), type(O2,plate).
```



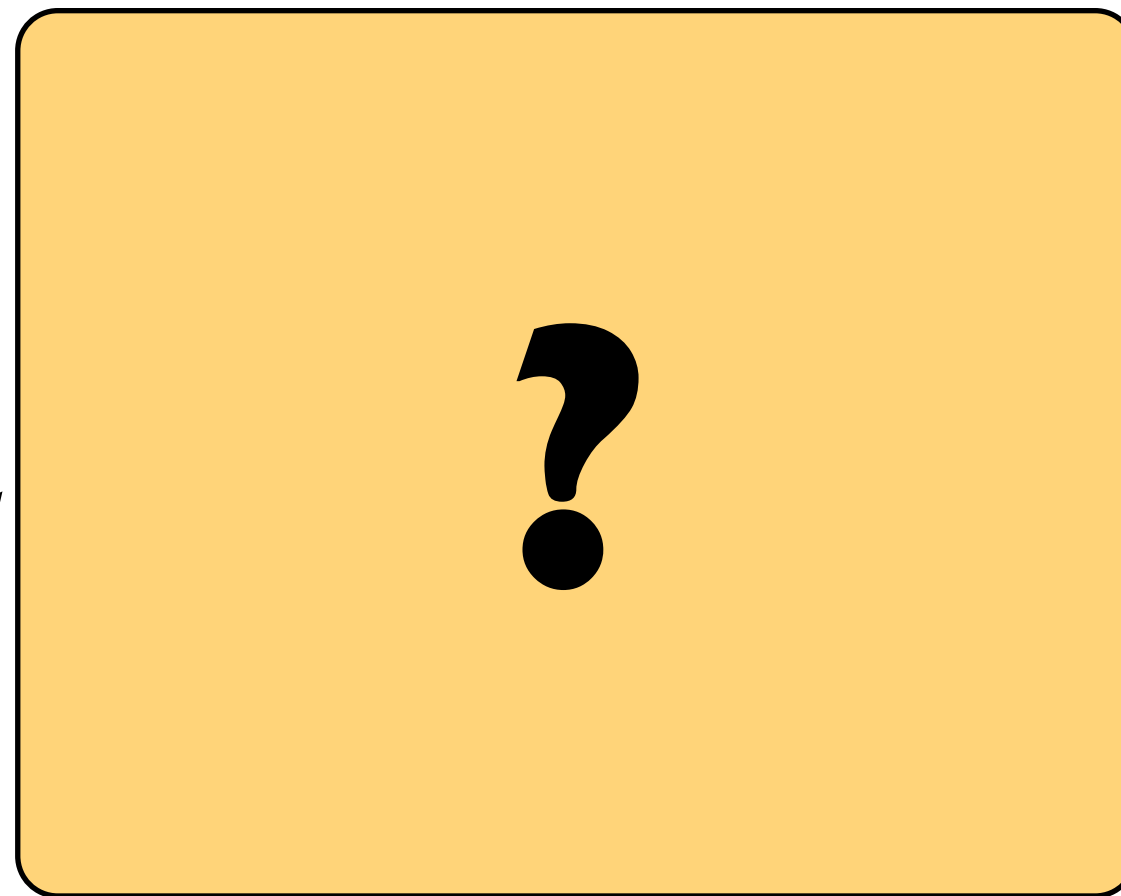
Answering Questions

Given:

program

queries

evidence



Find:

marginal
probabilities

conditional
probabilities

MPE state

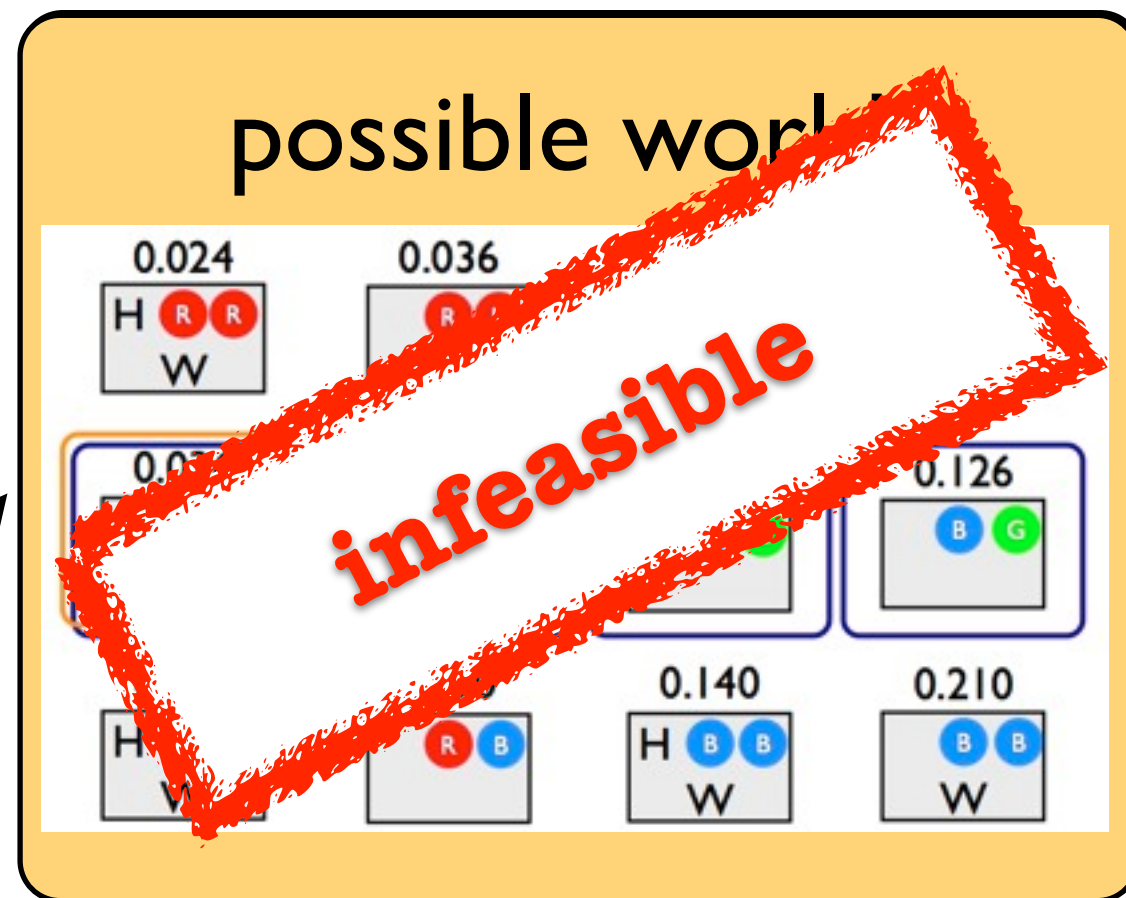
Answering Questions

Given:

program

queries

evidence



Find:

marginal probabilities

conditional probabilities

MPE state

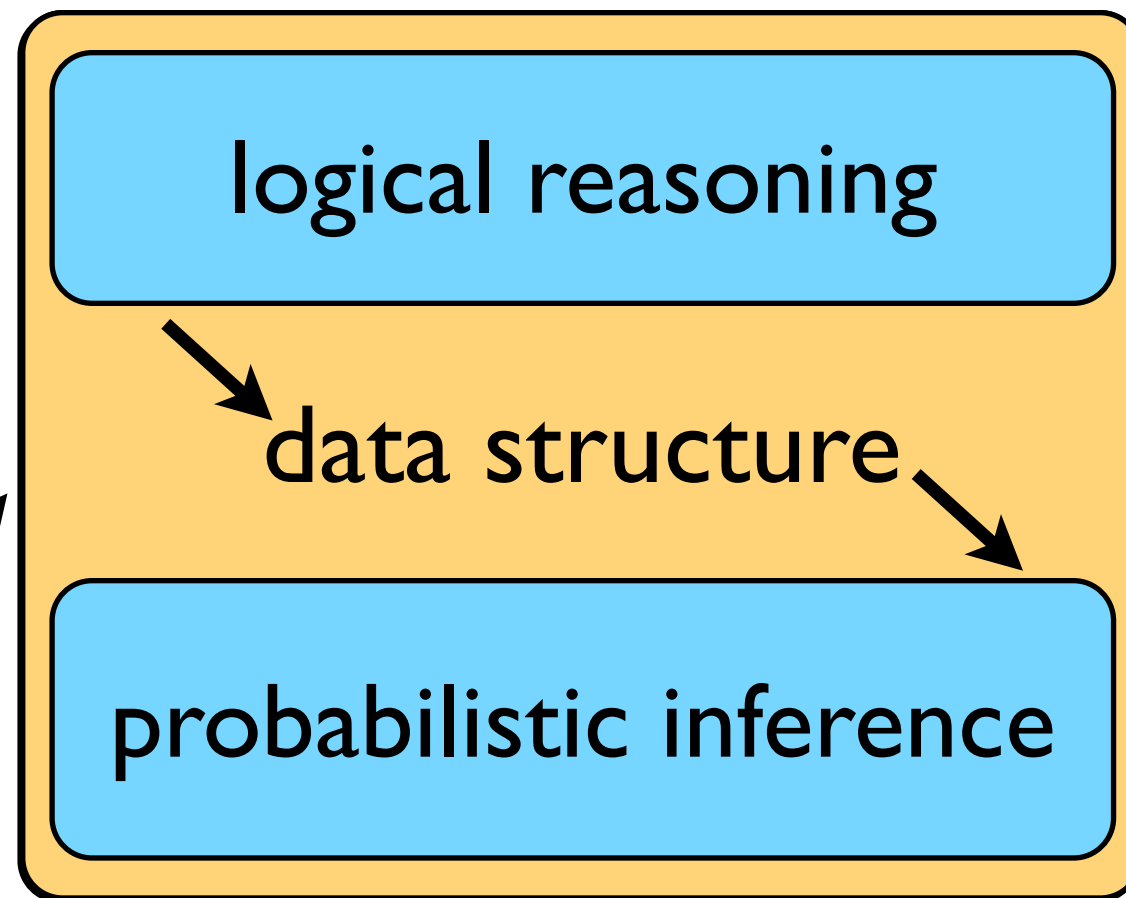
Answering Questions

Given:

program

queries

evidence



Find:

marginal
probabilities

conditional
probabilities

MPE state

Answering Questions

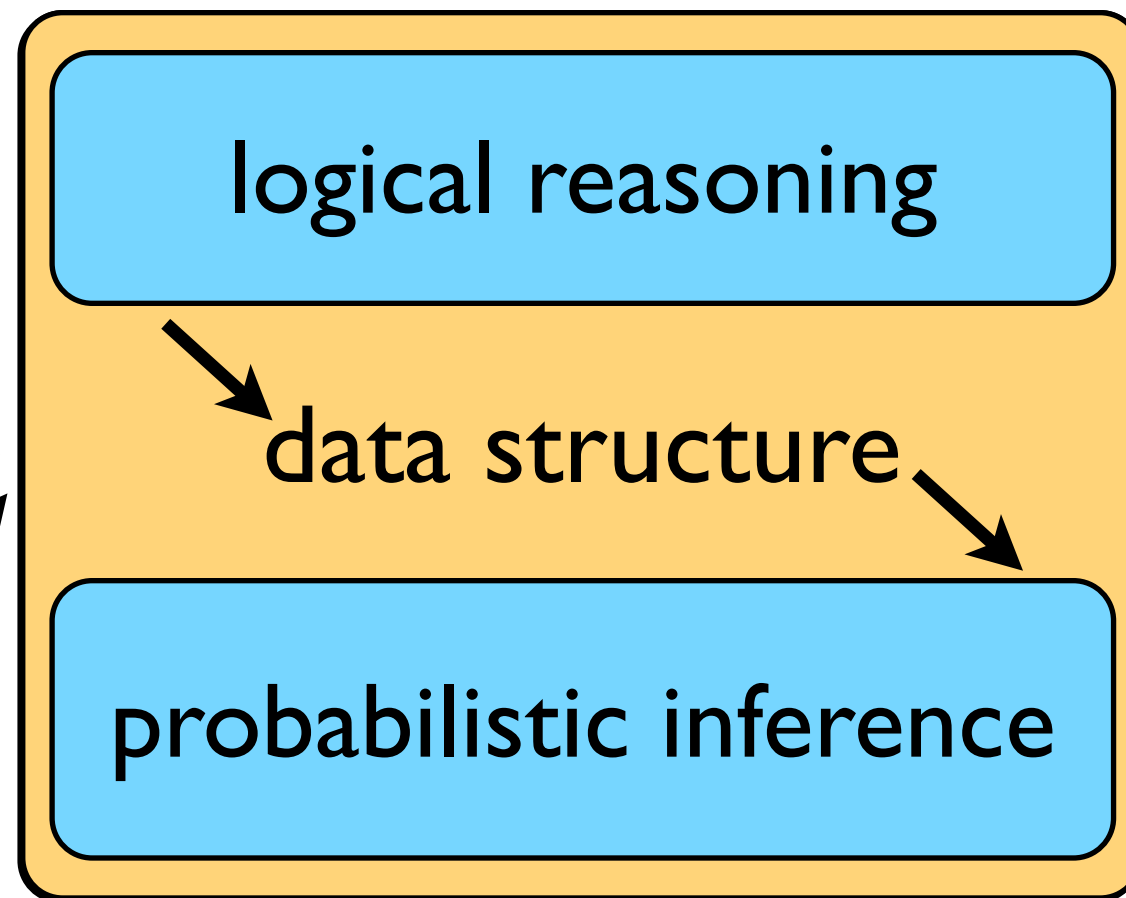
more on this later

Given:

program

queries

evidence



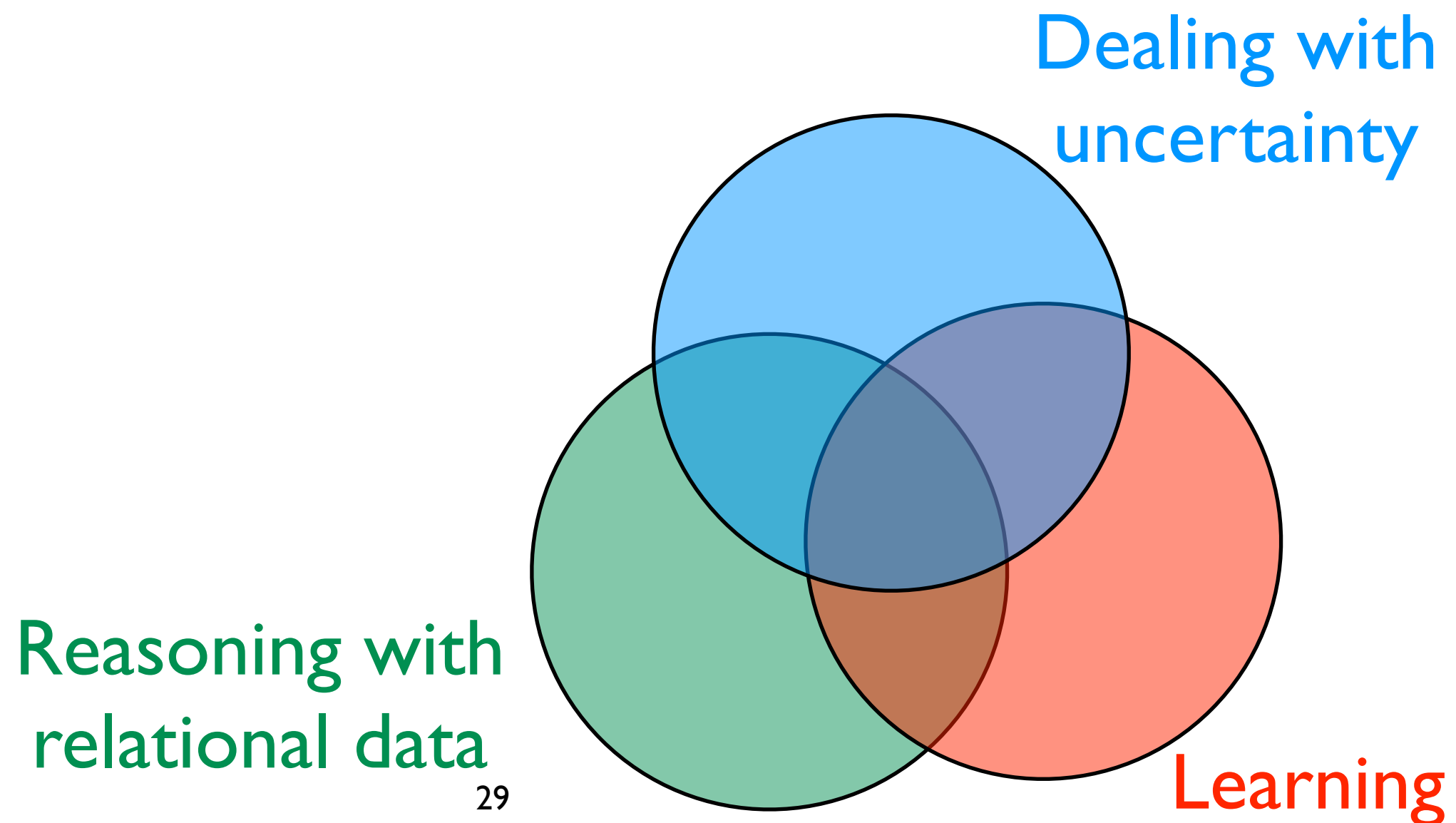
Find:

marginal
probabilities

conditional
probabilities

MPE state

Probabilistic Databases



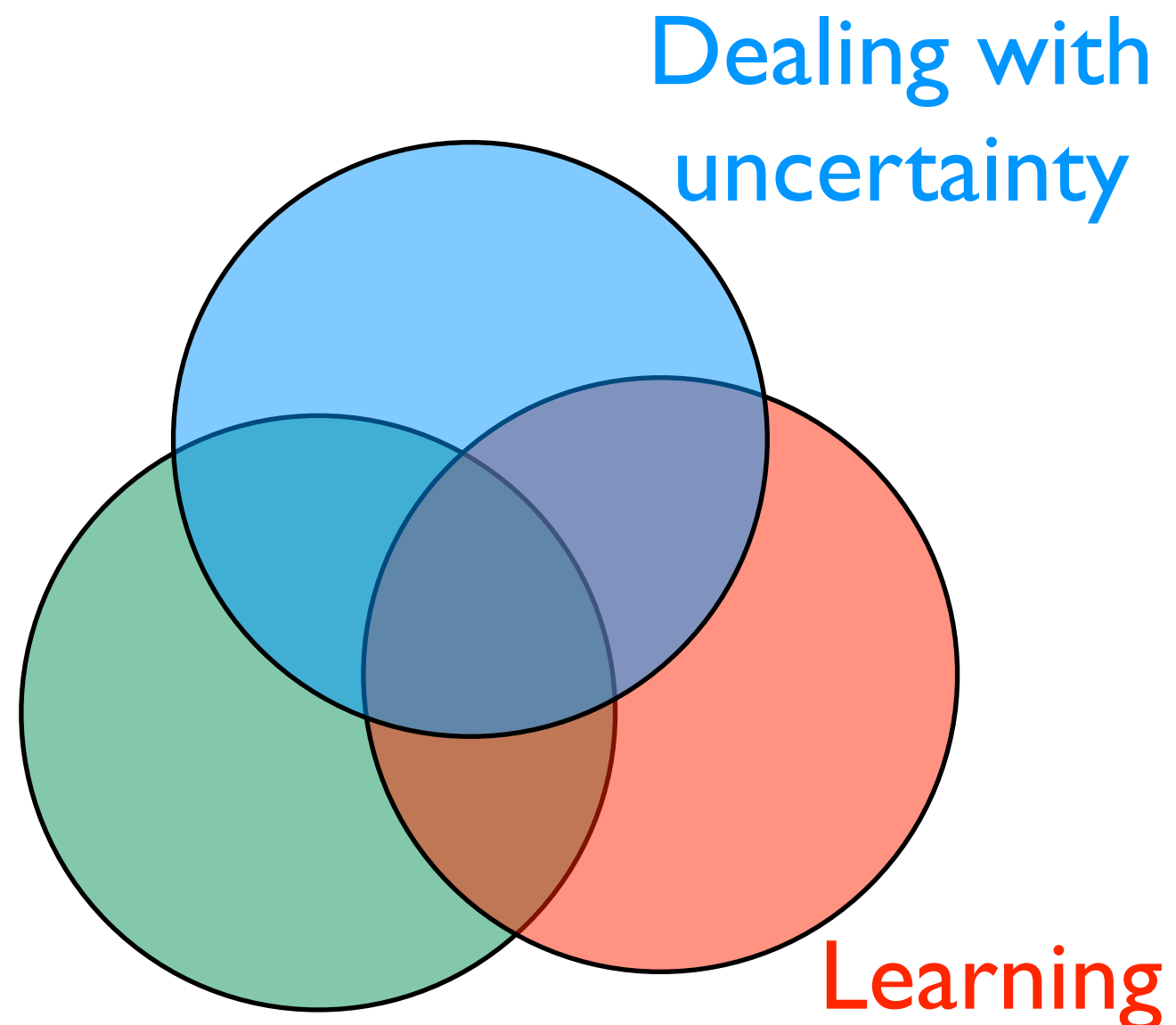
Probabilistic Databases

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

bornIn	
person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn	
city	country
london	uk
york	uk
paris	usa

relational
database



Probabilistic Databases

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

bornIn

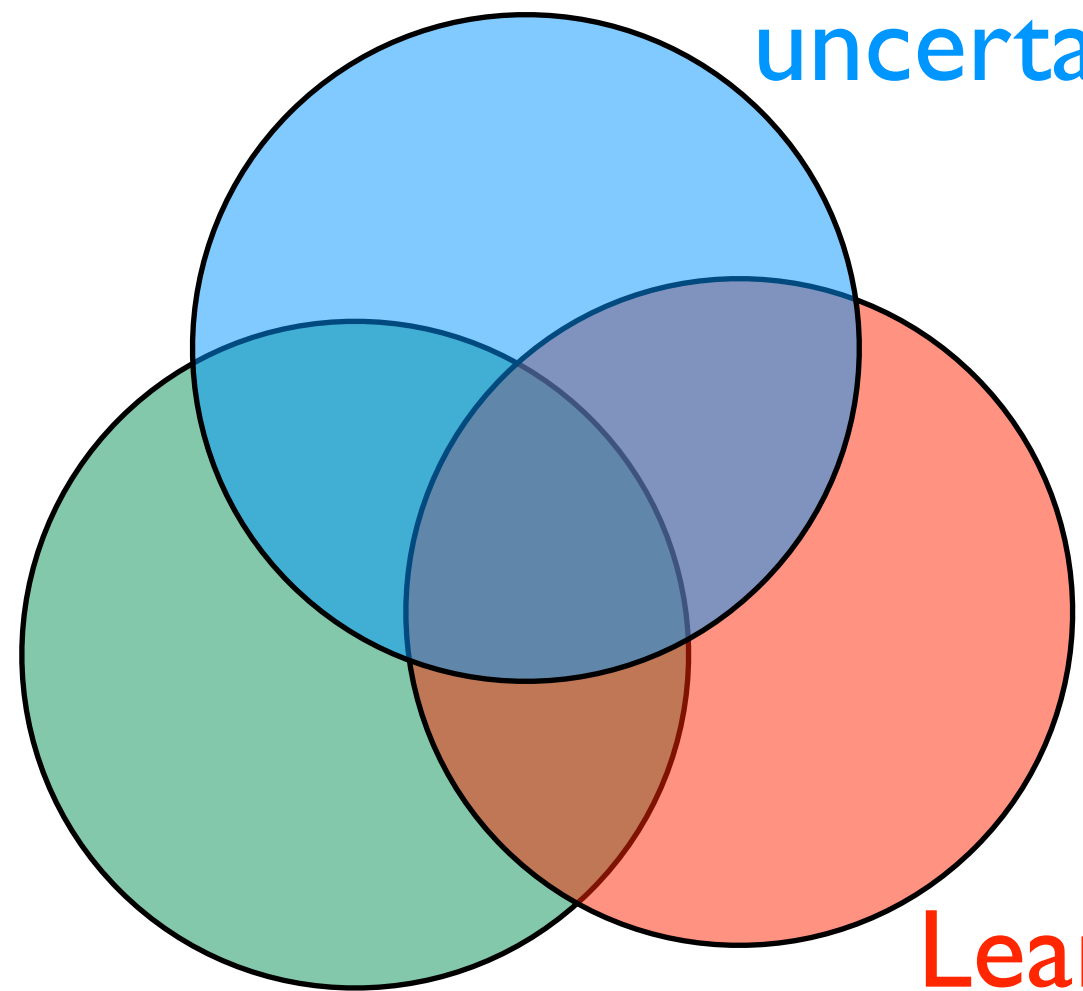
person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational
database

Dealing with
uncertainty



Learning

Probabilistic Databases

bornIn		
person	city	P
ann	london	0.87
bob	york	0.95
eve	new york	0.90
tom	paris	0.56

cityIn		
city	country	P
london	uk	0.99
york	uk	0.75
paris	usa	0.40

tuples as random
variables

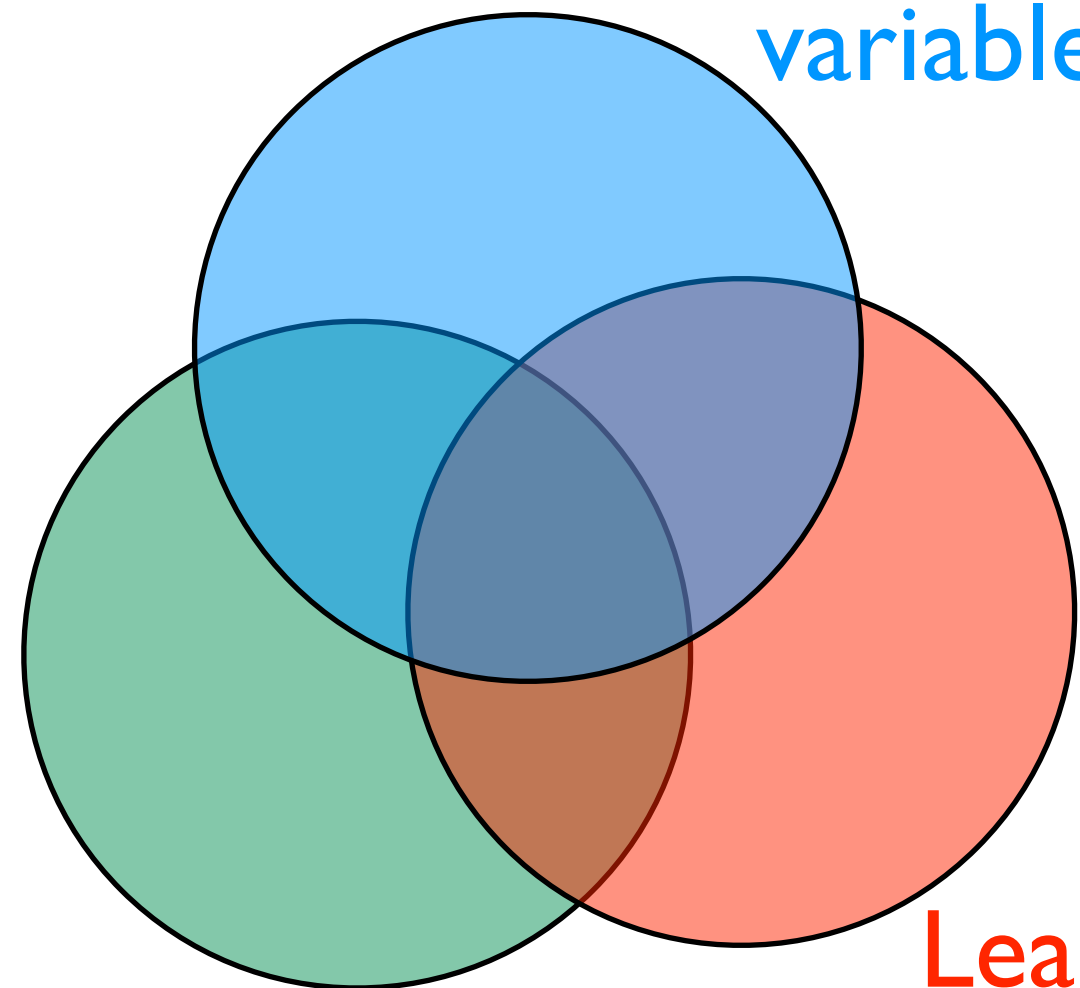
```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

bornIn	
person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn	
city	country
london	uk
york	uk
paris	usa

relational
database



Learning

Probabilistic Databases

several possible worlds

bornIn		
person	city	P
ann	london	0.87
bob	york	0.95
eve	new york	0.90
tom	paris	0.56

cityIn		
city	country	P
london	uk	0.99
york	uk	0.75
paris	usa	0.40

tuples as random

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

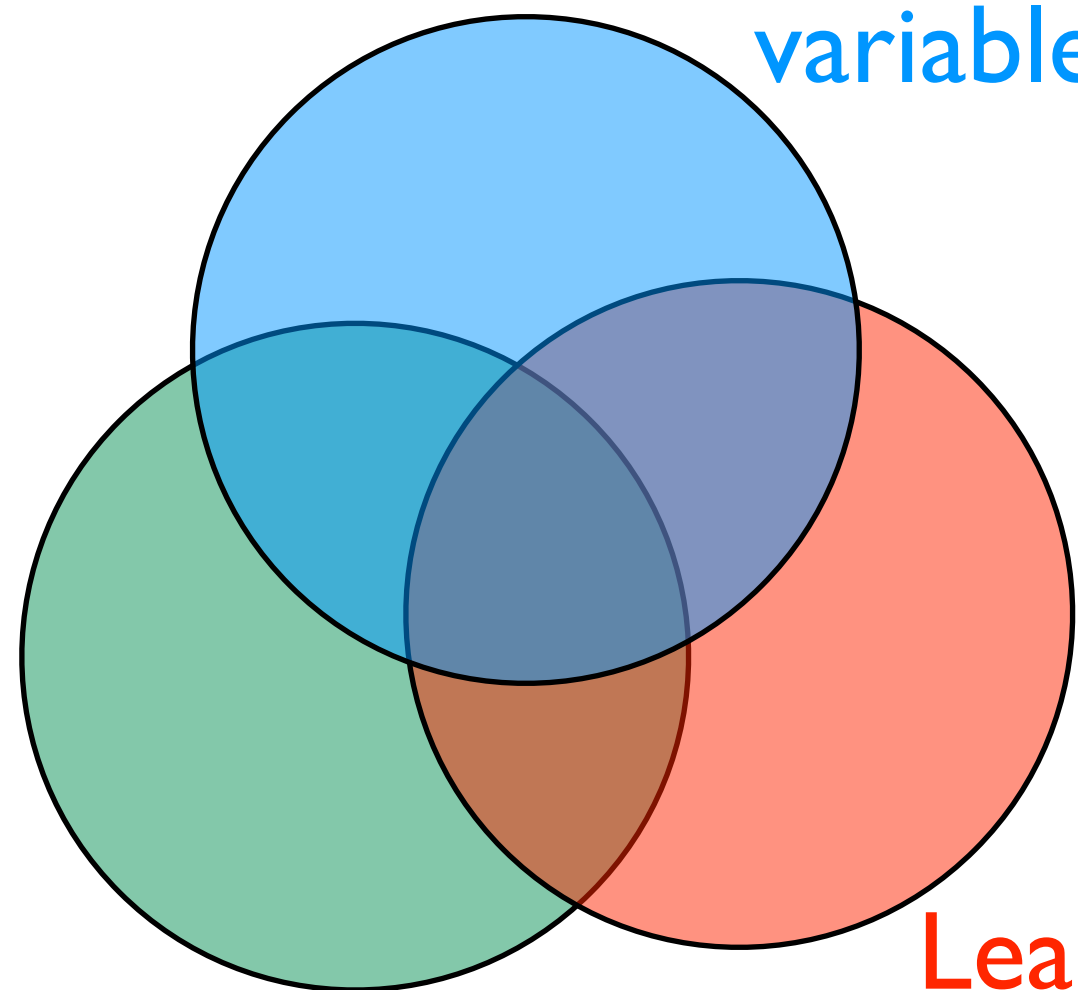
variables

one world

bornIn	
person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn	
city	country
london	uk
york	uk
paris	usa

relational
database



Learning

Probabilistic Databases

several possible worlds

bornIn		
person	city	P
ann	london	0.87
bob	york	0.95
		0.90
		0.56

cityIn		
city	country	P
london	uk	0.99
york	uk	0.75
paris	usa	0.40

probabilistic tables + database queries
→ distribution over possible worlds

tuples as random

variables

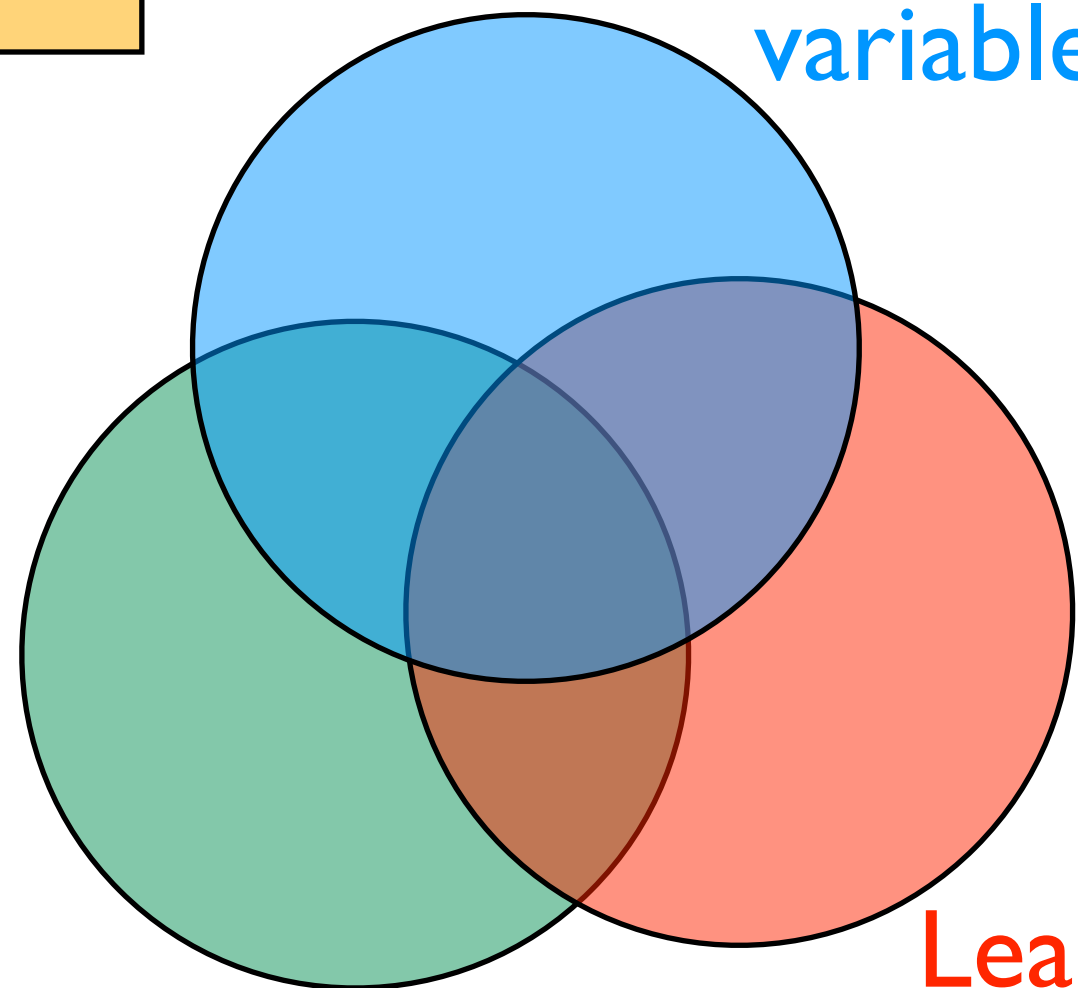
```
select
from bornIn x, cityIn y
where x.city=y.city
```

one world

bornIn	
person	city
ann	london
bob	york
eve	new york
tom	paris


cityIn	
city	country
london	uk
york	uk
paris	usa


relational
database























Learning

Example: Information Extraction

Recently-Learned Facts 



instance	iteration	date learned	confidence
<u>kelly andrews</u> is a <u>female</u>	826	29-mar-2014	98.7  
<u>investment next year</u> is an <u>economic sector</u>	829	10-apr-2014	95.3  
<u>shibenik</u> is a <u>geopolitical entity</u> that is an organization	829	10-apr-2014	97.2  
<u>quality web design work</u> is a <u>character trait</u>	826	29-mar-2014	91.0  
<u>mercedes benz cls by carlsson</u> is an <u>automobile manufacturer</u>	829	10-apr-2014	95.2  
<u>social work</u> is an academic program <u>at the university rutgers university</u>	827	02-apr-2014	93.8  
<u>dante wrote</u> the book <u>the divine comedy</u>	826	29-mar-2014	93.8  
<u>willie aames</u> was <u>born in</u> the city <u>los angeles</u>	831	16-apr-2014	100.0  
<u>kitt peak</u> is a mountain <u>in the state or province arizona</u>	831	16-apr-2014	96.9  
<u>greenwich</u> is a park <u>in the city london</u>	831	16-apr-2014	100.0  

↑
instances for many
different relations

↑
degree of certainty

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```


Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company
personal_computer	ibm
adobe_indesign	adobe
adobe_dreamweaver	adobe

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company
personal_computer	ibm
adobe_indesign	adobe
adobe_dreamweaver	adobe

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company
personal computer	ibm
adobe_indesign	adobe
adobe_dreamweaver	adobe

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company
personal_computer	ibm
adobe_indesign	adobe
adobe_dreamweaver	adobe

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

same query -
probabilities handled implicitly

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

$$0.96 \times 0.99 = 0.95$$

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

$$0.9 \times 0.93 = 0.83$$

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

$$0.87 \times 0.93 = 0.80$$

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

answer tuples ranked by
probability

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

PDB with tuple-level uncertainty in ProbLog?

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

PDB with tuple-level uncertainty in ProbLog?

ProducesProduct		
Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

PDB with tuple-level uncertainty in ProbLog?

ProducesProduct		
Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

```
0.96::producesProduct(sony,walkman) .
0.96::producesProduct(microsoft,mac_os_x) .
0.96::producesProduct(ibm,personal_computer) .
0.9::producesProduct(microsoft,mac_os) .
0.9::producesProduct(adobe,adobe_indesign) .
0.87::producesProduct(adobe,adobe_dreamweaver) .
...
```

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company  
from ProducesProduct x, HeadquarteredIn y  
where x.Company=y.Company and y.City='san_jose'
```

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company  
from ProducesProduct x, HeadquarteredIn y  
where x.Company=y.Company and y.City='san_jose'
```

```
result(Product, Company) :-  
    producesProduct(Company, Product),  
    headquarteredIn(Company, san_jose).  
query(result(_, _)).
```

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company  
from ProducesProduct x, HeadquarteredIn y  
where x.Company=y.Company and y.City='san_jose'
```

```
result(Product, Company) :-  
    producesProduct(Company, Product),  
    headquarteredIn(Company, san_jose).  
query(result(_, _)).
```

PDB with tuple-level uncertainty in ProbLog?

```
0.96::producesProduct(sony,walkman).
0.96::producesProduct(microsoft,mac_os_x).
0.96::producesProduct(ibm,personal_computer).
0.9::producesProduct(microsoft,mac_os).
0.9::producesProduct(adobe,adobe_indesign).
0.87::producesProduct(adobe,adobe_dreamweaver).
...
```

```
1.00::headquarteredIn(microsoft,redmond).
0.99::headquarteredIn(ibm,san_jose).
0.93::headquarteredIn(emirates_airlines,dubai).
0.93::headquarteredIn(honda,torrance).
0.93::headquarteredIn(horizon,seattle).
0.93::headquarteredIn(egyptair,cairo).
0.93::headquarteredIn(adobe,san_jose).
...
```

```
result(Product,Company) :- producesProduct(Company,Product),
                             headquarteredIn(Company,san_jose).
query(result(_,_) ).
```

PDB with attribute-level uncertainty in ProbLog?

color		
item	color	P
mug	green	0.65
	blue	0.35
plate	pink	0.23
	red	0.14
	purple	0.63

PDB with attribute-level uncertainty in ProbLog?

color		
item	color	P
mug	green	0.65
	blue	0.35
plate	pink	0.23
	red	0.14
	purple	0.63

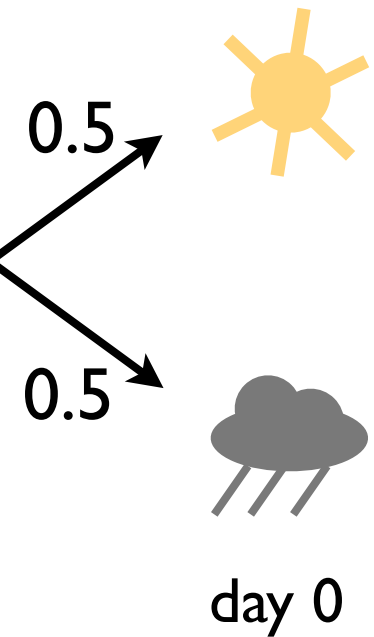
```
0.65::color(mug,green) ; 0.35::color(mug,blue) <- true.  
0.23::color(plate,pink) ; 0.14::color(plate,red) ;  
                                0.63::color(plate,purple) <- true.
```

ProbLog by example:

Rain or sun?

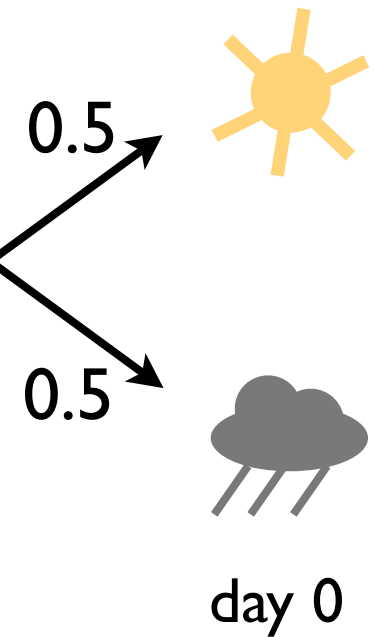
ProbLog by example:

Rain or sun?



ProbLog by example:

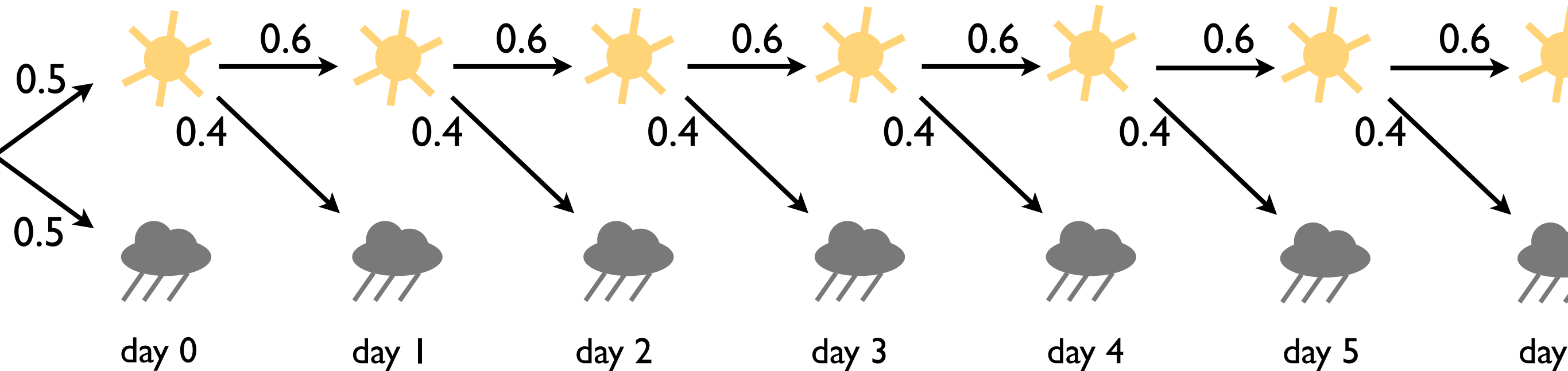
Rain or sun?



```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

ProbLog by example:

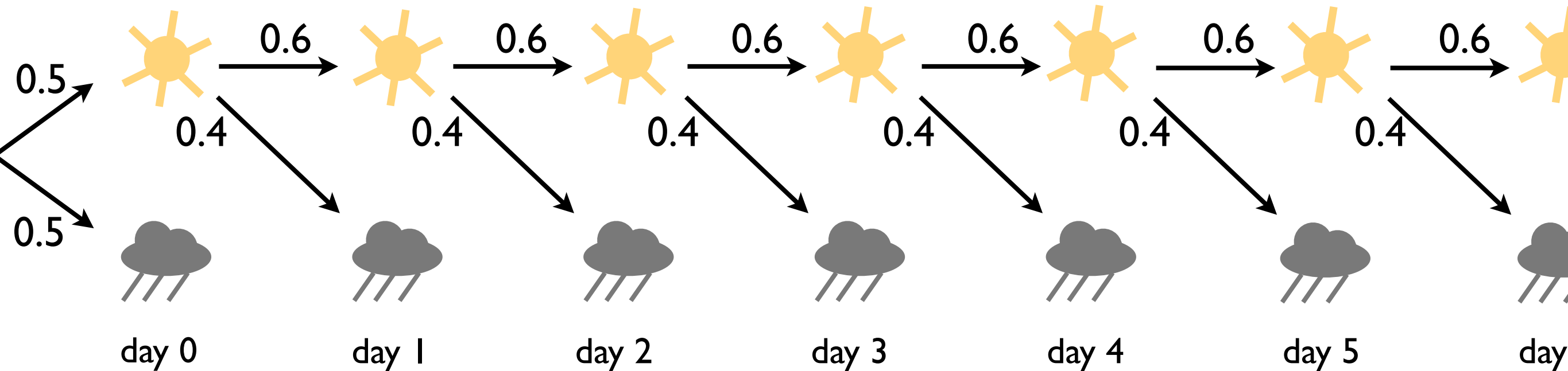
Rain or sun?



```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

ProbLog by example:

Rain or sun?

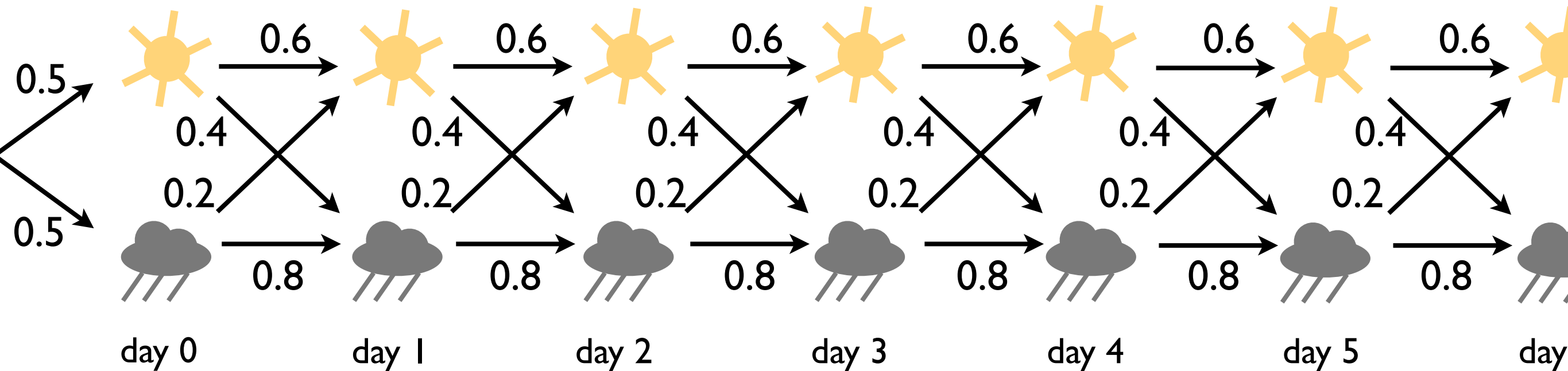


```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

```
0.6::weather(sun,T) ; 0.4::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(sun,Tprev) .
```

ProbLog by example:

Rain or sun?

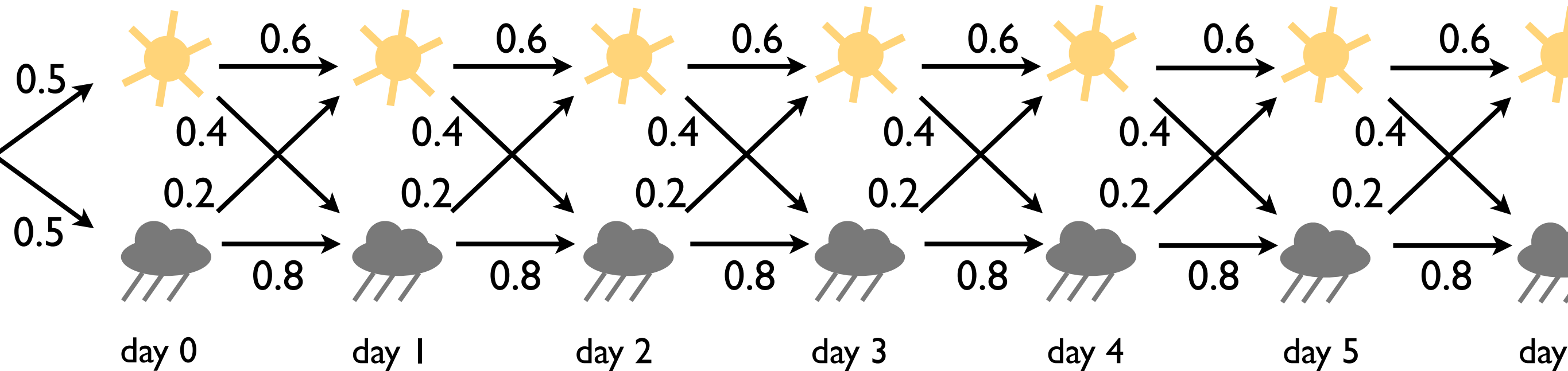


```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

```
0.6::weather(sun,T) ; 0.4::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(sun,Tprev) .
```

ProbLog by example:

Rain or sun?



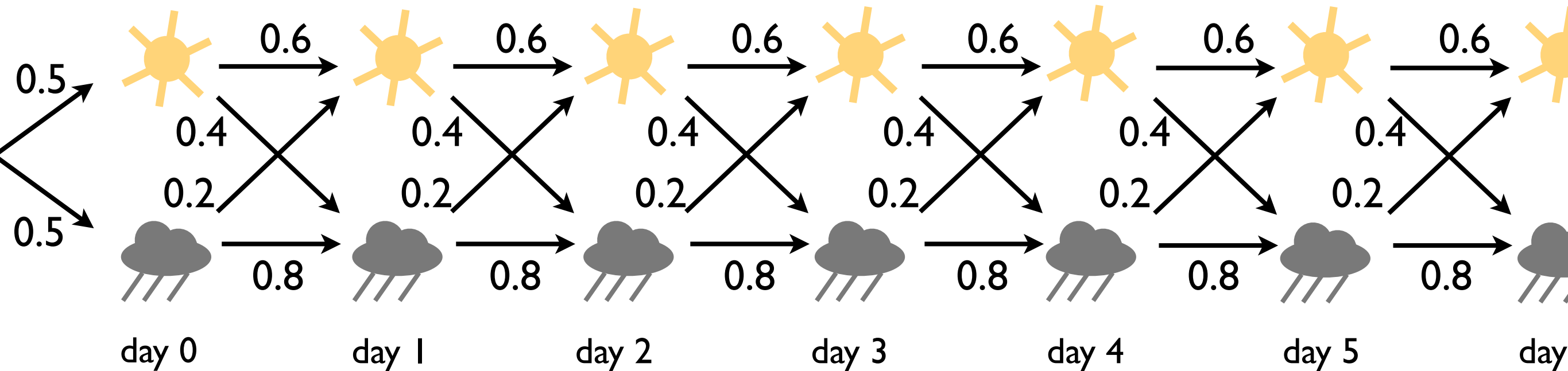
```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

```
0.6::weather(sun,T) ; 0.4::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(sun,Tprev) .
```

```
0.2::weather(sun,T) ; 0.8::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(rain,Tprev) .
```

ProbLog by example:

Rain or sun?



```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

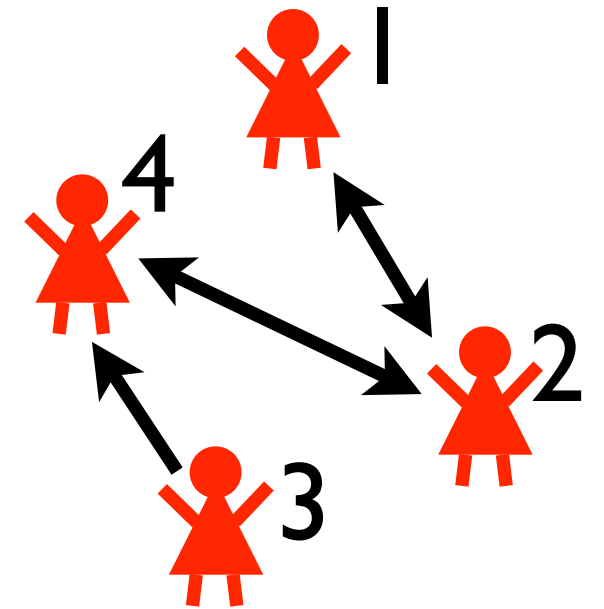
```
0.6::weather(sun,T) ; 0.4::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(sun,Tprev) .
```

```
0.2::weather(sun,T) ; 0.8::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(rain,Tprev) .
```

infinite possible worlds! BUT: finitely many partial worlds suffice to answer any given ground query

ProbLog by example:

Friends & smokers



```
person(1).  
person(2).  
person(3).  
person(4).
```

```
friend(1,2).  
friend(2,1).  
friend(2,4).  
friend(3,4).  
friend(4,2).
```

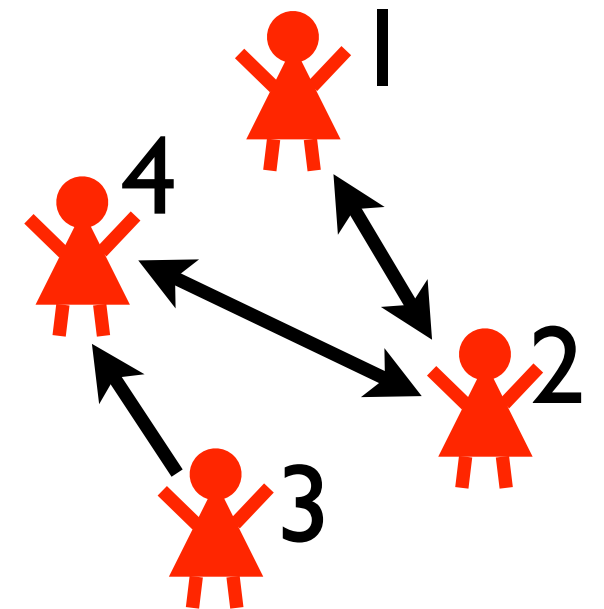

ProbLog by example:

Friends & smokers

typed probabilistic facts

= a probabilistic fact for each grounding

```
0.3::stress(X):- person(X).  
0.2::influences(X,Y):-  
    person(X), person(Y).
```



```
person(1).  
person(2).  
person(3).  
person(4).
```

```
friend(1,2).  
friend(2,1).  
friend(2,4).  
friend(3,4).  
friend(4,2).
```

ProbLog by example:

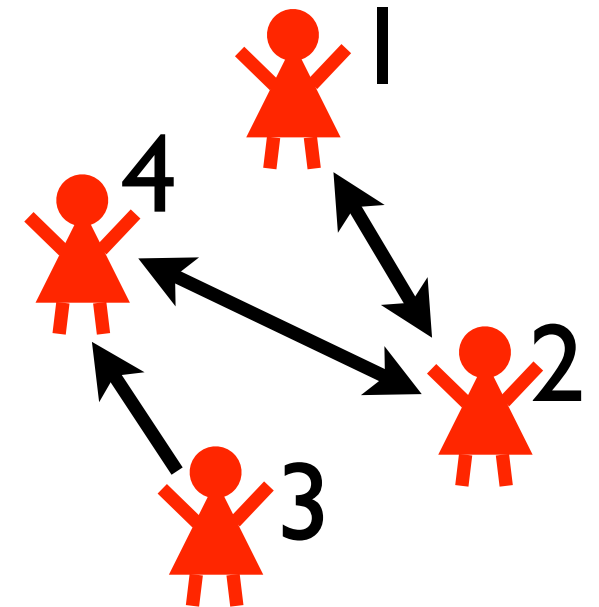
Friends & smokers

typed probabilistic facts

= a probabilistic fact for each grounding

```
0.3::stress(X):- person(X).  
0.2::influences(X,Y):-  
    person(X), person(Y).
```

```
0.3::stress(1).  
0.3::stress(2).  
0.3::stress(3).  
0.3::stress(4).  
  
0.2::influences(1,1).  
0.2::influences(1,2).  
0.2::influences(1,3).  
0.2::influences(1,4).  
0.2::influences(2,1).  
...  
0.2::influences(4,2).  
0.2::influences(4,3).  
0.2::influences(4,4).
```

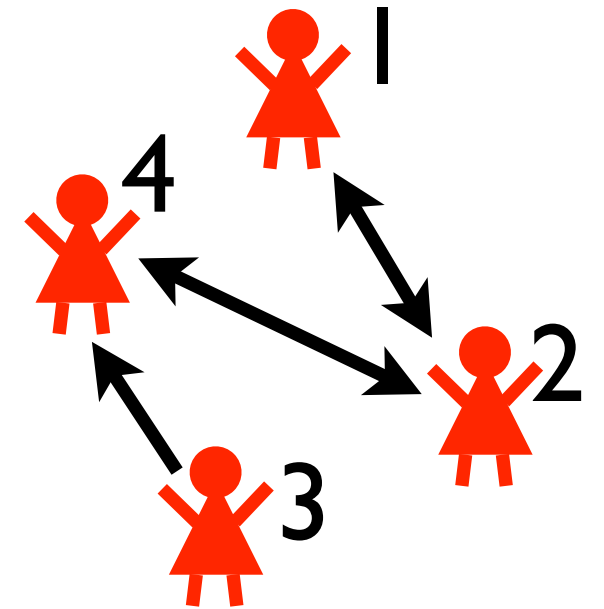


```
person(1).  
person(2).  
person(3).  
person(4).
```

```
friend(1,2).  
friend(2,1).  
friend(2,4).  
friend(3,4).  
friend(4,2).
```

ProbLog by example:

Friends & smokers



```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
    person(X), person(Y).

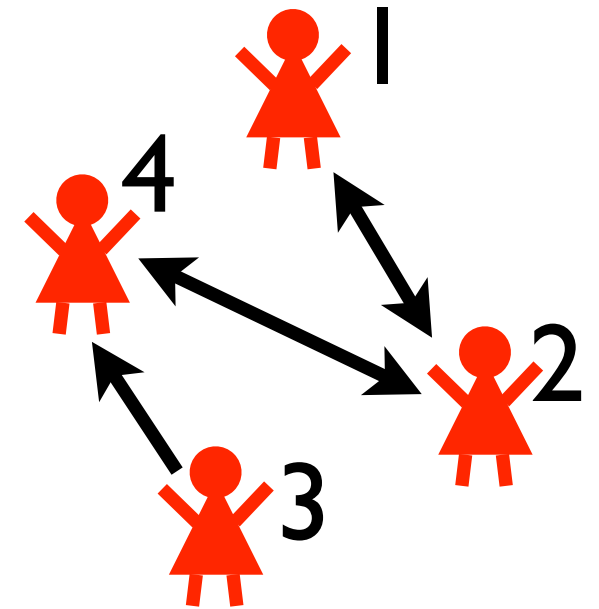
smokes(X) :- stress(X).
smokes(X) :-
    friend(X,Y), influences(Y,X), smokes(Y).
```

```
person(1).
person(2).
person(3).
person(4).
```

```
friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers



```
0.3::stress(X) :- person(X) .
```

```
0.2::influences(X,Y) :-  
    person(X) , person(Y) .
```

```
smokes(X) :- stress(X) .
```

```
smokes(X) :-  
    friend(X,Y) , influences(Y,X) , smokes(Y) .
```

```
0.4::asthma(X) <- smokes(X) .
```

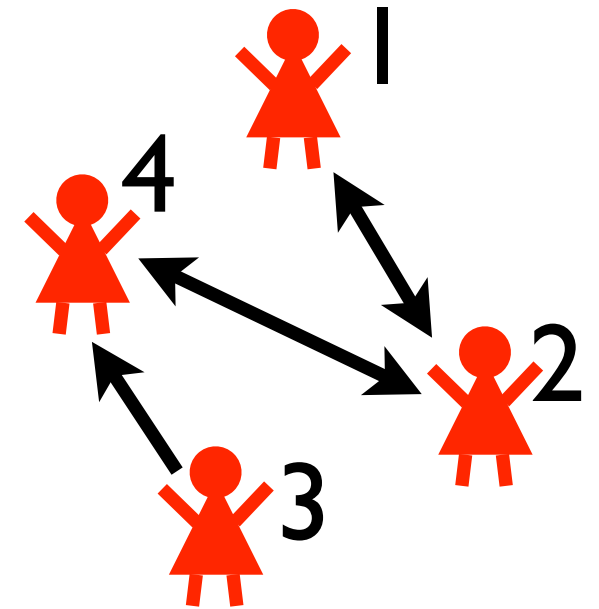
```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

annotated disjunction with implicit head atom:
with probability 0.6, nothing happens

ProbLog by example:

Friends & smokers



```
0.3::stress(X) :- person(X) .
0.2::influences(X,Y) :-
    person(X) , person(Y) .

smokes(X) :- stress(X) .
smokes(X) :-
    friend(X,Y) , influences(Y,X) , smokes(Y) .

0.4::asthma(X) <- smokes(X) .
```

```
person(1) .
person(2) .
person(3) .
person(4) .
```

```
friend(1,2) .
friend(2,1) .
friend(2,4) .
friend(3,4) .
friend(4,2) .
```

ProbLog by example:

Limited Luggage



```
weight(skis,6) .  
weight(boots,4) .  
weight(helmet,3) .  
weight(gloves,2) .
```

ProbLog by example:

Limited Luggage



```
weight(skis,6) .  
weight(boots,4) .  
weight(helmet,3) .  
weight(gloves,2) .
```

```
P::pack(Item) :- weight(Item,Weight) , P is 1.0/Weight.
```

flexible probability:
computed from the weight of the item

ProbLog by example:

Limited Luggage



<code>weight(skis,6) .</code>	<code>1/6::pack(skis) .</code>
<code>weight(boots,4) .</code>	<code>1/4::pack(boots) .</code>
<code>weight(helmet,3) .</code>	<code>1/3::pack(helmet) .</code>
<code>weight(gloves,2) .</code>	<code>1/2::pack(gloves) .</code>

`P::pack(Item) :- weight(Item,Weight) , P is 1.0/Weight.`

flexible probability:
computed from the weight of the item

ProbLog by example:

Limited Luggage



```
weight(skis,6) .  
weight(boots,4) .  
weight(helmet,3) .  
weight(gloves,2) .
```

```
P::pack(Item) :- weight(Item,Weight), P is 1.0/Weight.
```

```
excess(Limit) :- excess([skis,boots,helmet,gloves],Limit) .
```

list of all items

ProbLog by example:

Limited Luggage



```
weight(skis,6) .  
weight(boots,4) .  
weight(helmet,3) .  
weight(gloves,2) .
```

```
P::pack(Item) :- weight(Item,Weight), P is 1.0/Weight.
```

```
excess(Limit) :- excess([skis,boots,helmet,gloves],Limit) .
```

```
excess([],Limit) :- Limit<0.
```

```
excess([I|R],Limit) :-
```

```
    pack(I), weight(I,W), L is Limit-W, excess(R,L) .
```

```
excess([I|R],Limit) :-
```

```
    \+pack(I), excess(R,Limit) .
```

ProbLog by example:

Limited Luggage



```
weight(skis,6) .  
weight(boots,4) .  
weight(helmet,3) .  
weight(gloves,2) .
```

```
P::pack(Item) :- weight(Item,Weight), P is 1.0/Weight.
```

```
excess(Limit) :- excess([skis,boots,helmet,gloves],Limit) .
```

```
excess([],Limit) :- Limit<0.
```

```
excess([I|R],Limit) :-  
    pack(I), weight(I,W), L is Limit-W, excess(R,L) .
```

```
excess([I|R],Limit) .  
    \+pack(I), excess(R,Limit) .
```

pack first item, decrease
limit by its weight, and
continue with rest of items

ProbLog by example:

Limited Luggage



```
weight(skis,6) .  
weight(boots,4) .  
weight(helmet,3) .  
weight(gloves,2) .
```

```
P::pack(Item) :- weight(Item,Weight), P is 1.0/Weight.
```

```
excess(Limit) :- excess([skis,boots,helmet,gloves],Limit) .
```

```
excess([],Limit) :- Limit<0.
```

```
excess([I|R],Limit) :-
```

```
    pack(I), weight(I,W), L is Limit-W, excess(R,L) .
```

```
excess([I|R],Limit) :-
```

```
    \+pack(I), excess(R,Limit) .
```

do **not** pack first item,
continue with rest of items

ProbLog by example:

Limited Luggage



```
weight(skis,6) .  
weight(boots,4) .  
weight(helmet,3) .  
weight(gloves,2) .
```

```
P::pack(Item) :- weight(Item,Weight), P is 1.0/Weight.
```

```
excess(Limit) :- excess([skis,boots,helmet,gloves],Limit) .
```

```
excess([],Limit) :- Limit<0.
```

```
excess([I|R],Limit) :-
```

```
    pack(I), weight(I,W), L is Limit-W, excess(R,L) .
```

```
excess([I|R],Limit) :-
```

```
    \+pack(I), excess(R,Limit) .
```

no items left: did we add too much?

ProbLog by example:

Limited Luggage



```
weight(skis,6) .  
weight(boots,4) .  
weight(helmet,3) .  
weight(gloves,2) .
```

```
P::pack(Item) :- weight(Item,Weight), P is 1.0/Weight.
```

```
excess(Limit) :- excess([skis,boots,helmet,gloves],Limit) .
```

```
excess([],Limit) :- Limit<0.
```

```
excess([I|R],Limit) :-
```

```
    pack(I), weight(I,W), L is Limit-W, excess(R,L) .
```

```
excess([I|R],Limit) :-
```

```
    \+pack(I), excess(R,Limit) .
```

Summary: ProbLog Syntax

- input database: ground facts `person(bob) .`
- probabilistic facts `0.5::stress(bob) .`
- typed probabilistic facts
(body deterministic) `0.5::stress(X) :- person(X) .`
- flexible probabilities `P::pack(Item) :- weight(Item,W) ,
P is 1.0/W.`
- annotated disjunctions `0.4::asthma(X) <- smokes(X) .`
`0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.`
- Prolog clauses `smokes(X) :- influences(Y,X) , smokes(Y) .`
`excess([I|R],Limit) :- \+pack(I) , excess(R,Limit) .`

ProbLog example:

Friends & smokers

```
0.5::stress(1).
```

```
0.1::stress(2).
```

```
0.8::stress(3).
```

```
0.3::stress(4).
```

```
0.9::friend(1,2).
```

```
0.8::friend(2,1).
```

```
0.3::friend(2,4).
```

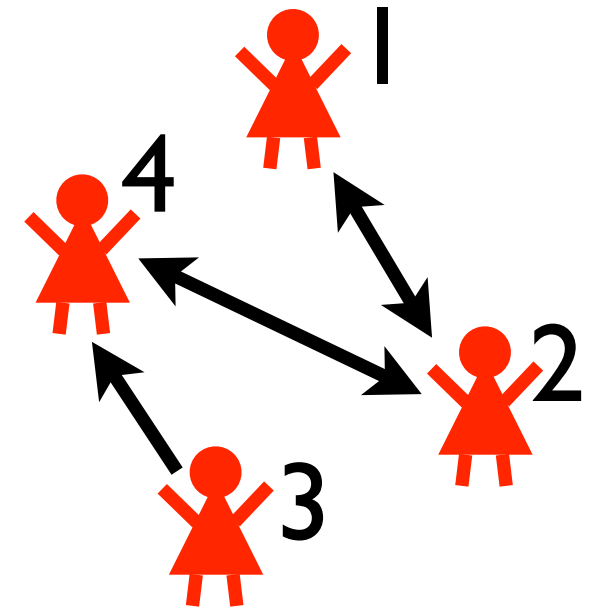
```
0.7::friend(3,4).
```

```
0.1::friend(4,2).
```

```
smokes(X) :- stress(X).
```

```
smokes(X) :-
```

```
    friend(Y,X), smokes(Y).
```



ProbLog example:

Friends & smokers

```
0.5::stress(1).
```

```
0.1::stress(2).
```

```
0.8::stress(3).
```

```
0.3::stress(4).
```

```
0.9::friend(1,2).
```

```
0.8::friend(2,1).
```

```
0.3::friend(2,4).
```

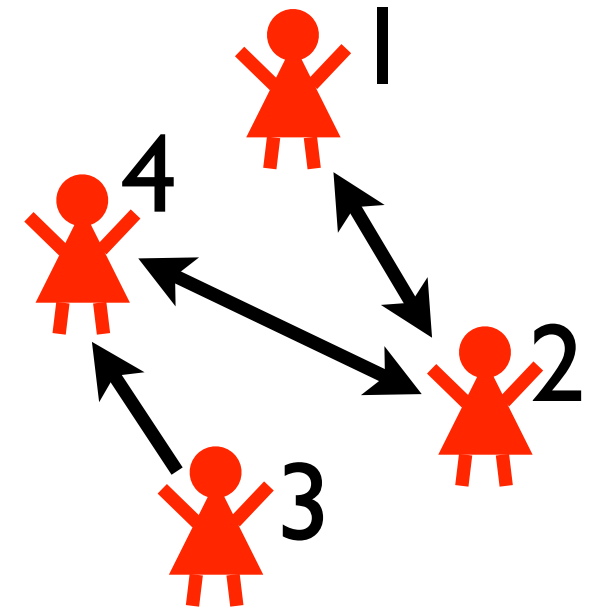
```
0.7::friend(3,4).
```

```
0.1::friend(4,2).
```

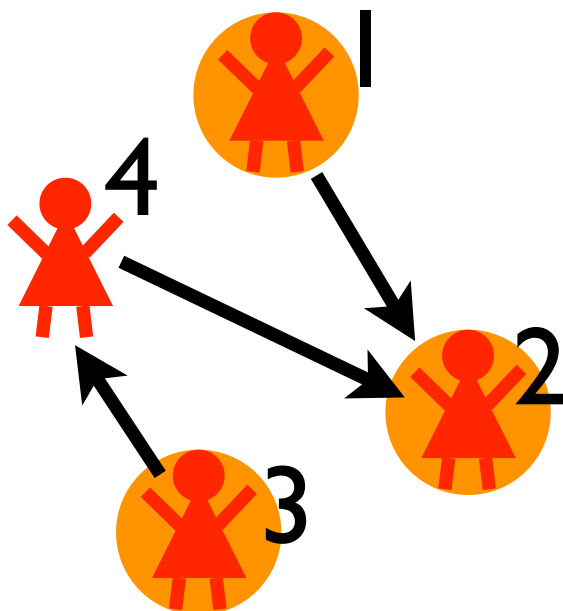
```
smokes(X) :- stress(X).
```

```
smokes(X) :-
```

```
    friend(Y,X), smokes(Y).
```



example possible world



```
friend(1,2).
```

```
friend(3,4).
```

```
friend(4,2).
```

```
stress(1).
```

```
stress(2).
```

```
stress(3).
```

```
smokes(1).
```

```
smokes(2).
```

```
smokes(3).
```

```
smokes(4).
```

ProbLog example:

Friends & smokers

```
0.5::stress(1).
```

```
0.1::stress(2).
```

```
0.8::stress(3).
```

```
0.3::stress(4).
```

```
0.9::friend(1,2).
```

```
0.8::friend(2,1).
```

```
0.3::friend(2,4).
```

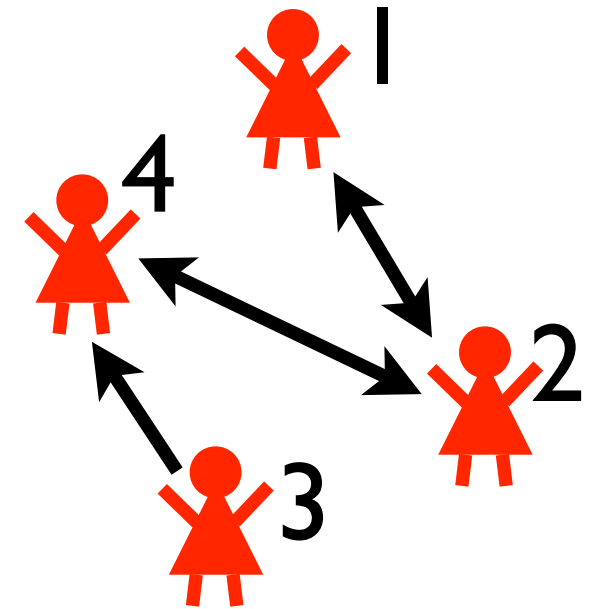
```
0.7::friend(3,4).
```

```
0.1::friend(4,2).
```

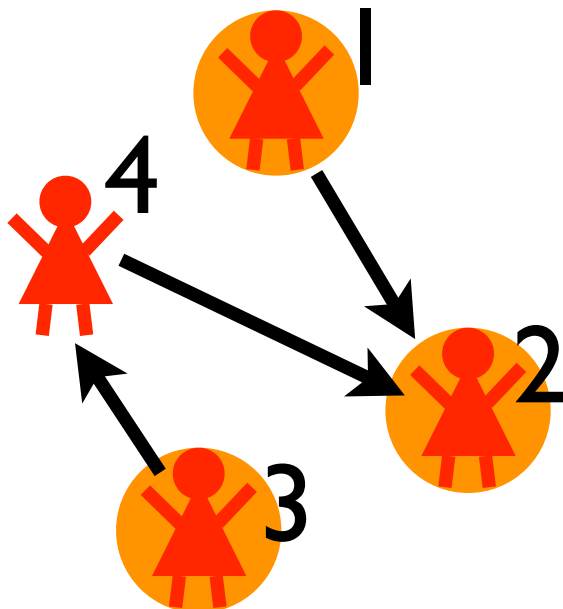
```
smokes(X) :- stress(X).
```

```
smokes(X) :-
```

```
    friend(Y,X), smokes(Y).
```



example possible world



```
friend(1,2).
```

```
friend(3,4).
```

```
friend(4,2).
```

```
stress(1).
```

```
stress(2).
```

```
stress(3).
```

```
smokes(1).
```

```
smokes(2).
```

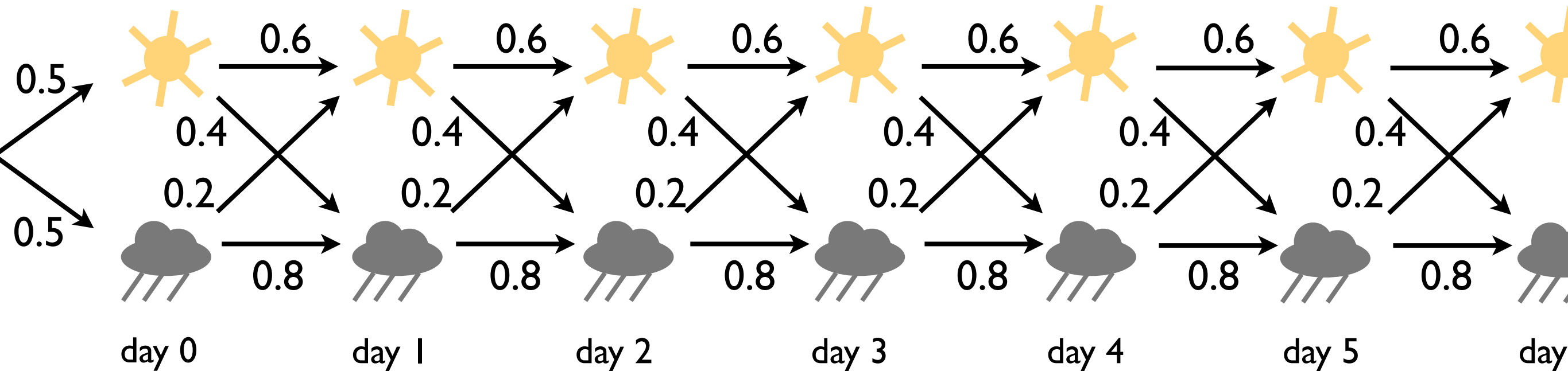
```
smokes(3).
```

```
smokes(4).
```

- several instances of **smokes (X)** in same world
- **smokes (2)** : multiple derivations in same world
- distribution over worlds not (always) a distribution over computations / answers

ProbLog by example:

Rain or sun?



```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

```
0.6::weather(sun,T) ; 0.4::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(sun,Tprev) .
```

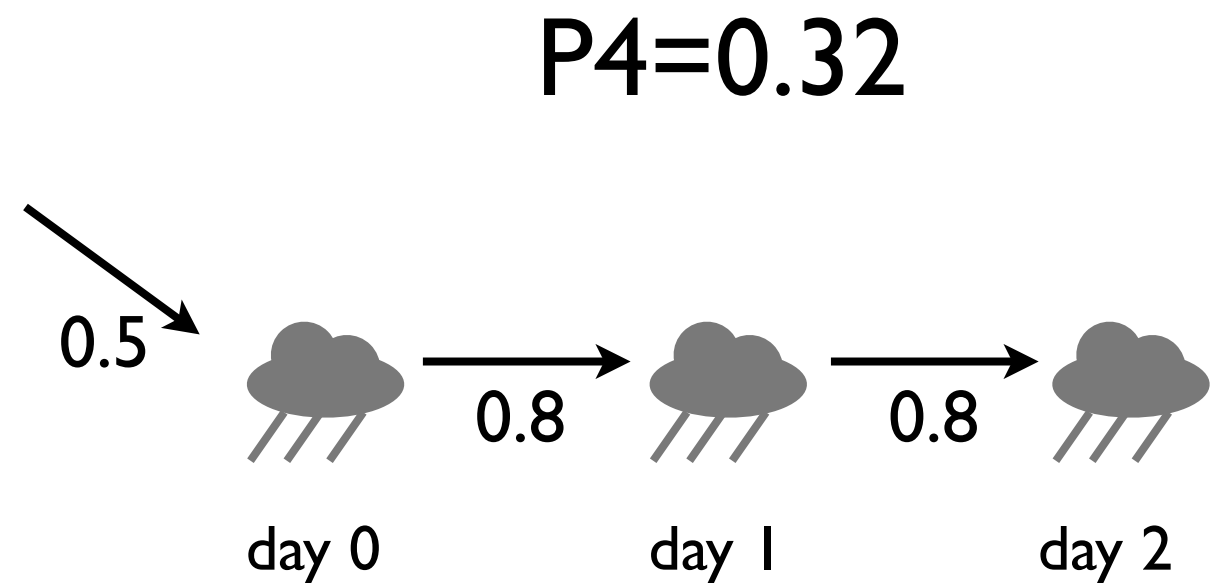
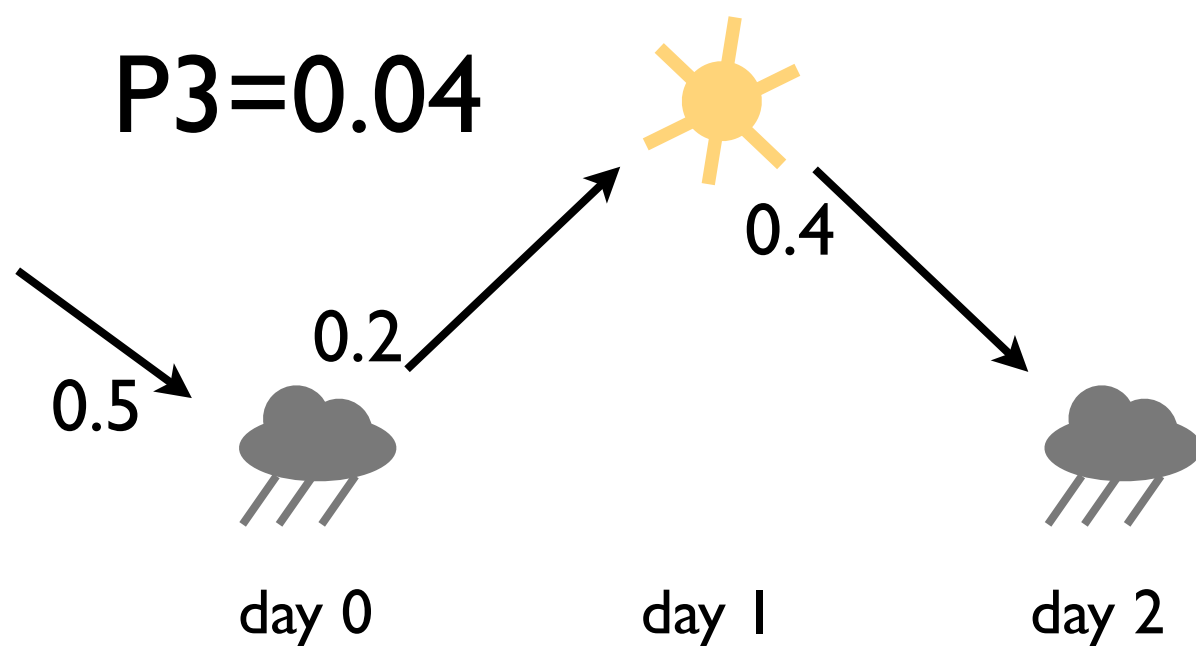
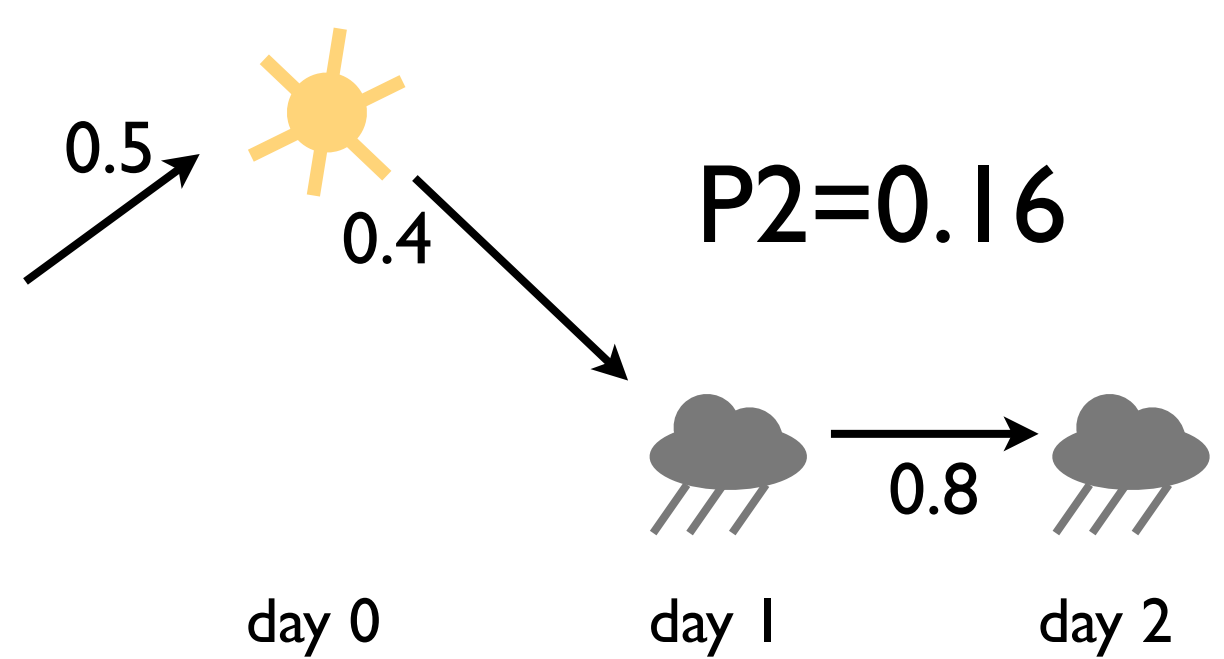
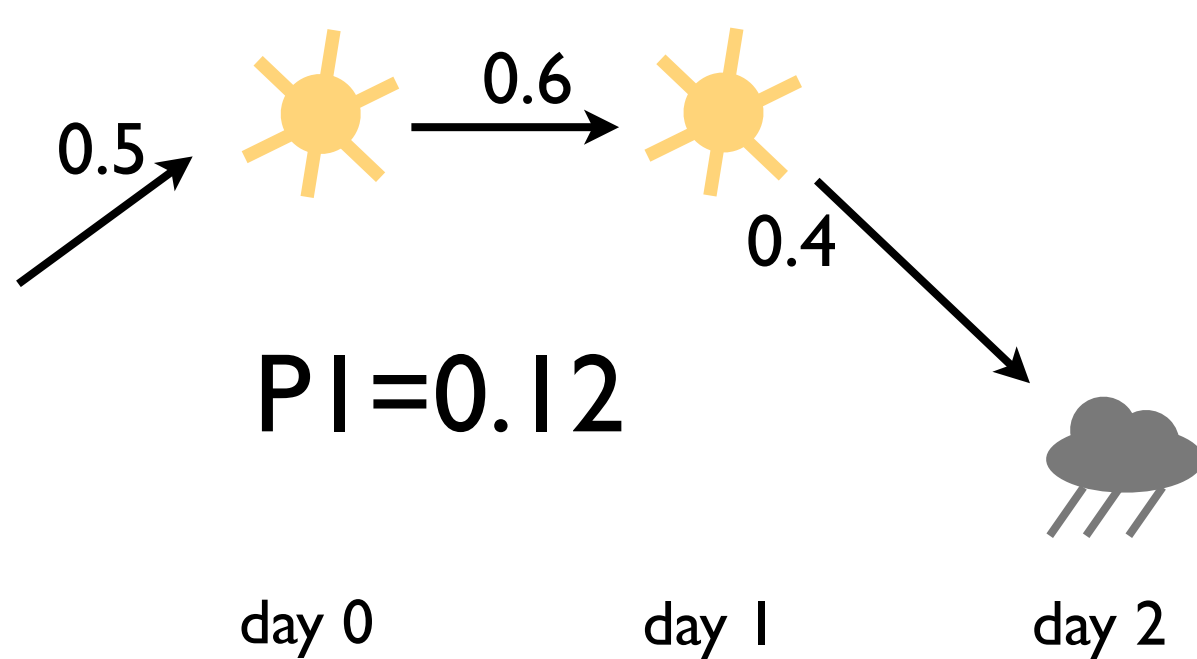
```
0.2::weather(sun,T) ; 0.8::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(rain,Tprev) .
```

\leq | proof for a ground query per possible world \rightarrow
distribution over worlds is distribution over derivations!

Possible worlds

?- weather(rain,2).

$$P = P1 + P2 + P3 + P4$$



Mutually Exclusive Rules:

no two rules apply simultaneously

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

```
0.6::weather(sun,T) ; 0.4::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(sun,Tprev) .
```

```
0.2::weather(sun,T) ; 0.8::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(rain,Tprev) .
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

```
0.6::weather(sun,T) ; 0.4::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(sun,Tprev) .
```

```
0.2::weather(sun,T) ; 0.8::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(rain,Tprev) .
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

day 0: either sun or rain

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

```
0.6::weather(sun,T) ; 0.4::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(sun,Tprev) .
```

```
0.2::weather(sun,T) ; 0.8::weather(rain,T)  
    <- T>0, Tprev is T-1, weather(rain,Tprev) .
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

day 0: either sun or rain

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

```
0.6::weather(sun,T) ; 0.4::weather(rain,T)  
  <- T>0, Tprev is T-1, weather(sun,Tprev) .
```

```
0.2::weather(sun,T) ; 0.8::weather(rain,T)  
  <- T>0, Tprev is T-1, weather(rain,Tprev) .
```

rules for $T > 0$ cover mutually exclusive cases
on previous day

PRISM

- Another probabilistic Prolog based on the distribution semantics
- Mutual exclusiveness assumption
 - allows for efficient inference by dynamic programming, cf. probabilistic grammars
 - but excludes certain models, e.g., smokers
- <http://sato-www.cs.titech.ac.jp/prism/>

PRISM

- “multi-valued random switches” = annotated disjunctions with body *true*
- switch gives fresh result on each call
- Prolog rules
- limited support for negation (compiling away)

Weather in PRISM

```
values (init, [sun, rain]) .  
values (tr(_), [sun, rain]) .
```

```
:- set_sw (init, [0.5, 0.5]) .  
:- set_sw (tr(sun), [0.6, 0.4]) .  
:- set_sw (tr(rain), [0.2, 0.8]) .
```

```
weather (W, Time) :-  
    Time >= 0,  
    msw (init, W0) ,  
    w (0, Time, W0, W) .
```

```
w (T, T, W, W) .  
w (Now, T, WNow, WT) :-  
    Now < T,  
    msw (tr (WNow), WNext) ,  
    Next is Now+1,  
    w (Next, T, WNext, WT) .
```

Weather in PRISM

```
values(init,[sun,rain]).  
values(tr(_),[sun,rain])
```

random variables and their values

```
:- set_sw(init,[0.5,0.5]).  
:- set_sw(tr(sun),[0.6,0.4]).  
:- set_sw(tr(rain),[0.2,0.8]).
```

```
weather(W,Time) :-  
    Time >= 0,  
    msw(init,W0),  
    w(0,Time,W0,W).
```

```
w(T,T,W,W).  
w(Now,T,WNow,WT) :-  
    Now < T,  
    msw(tr(WNow),WNext),  
    Next is Now+1,  
    w(Next,T,WNext,WT).
```

Weather in PRISM

```
values(init,[sun,rain]).  
values(tr(_),[sun,rain])
```

random variables and their values

```
:- set_sw(init,[0.5,0.5]).  
:- set_sw(tr(sun),[0.6,0.4]).  
:- set_sw(tr(rain),[0.2,0.8]).
```

probability distributions

```
weather(W,Time) :-  
    Time >= 0,  
    msw(init,W0),  
    w(0,Time,W0,W).
```

```
w(T,T,W,W).
```

```
w(Now,T,WNow,WT) :-  
    Now < T,  
    msw(tr(WNow),WNext),  
    Next is Now+1,  
    w(Next,T,WNext,WT).
```

Weather in PRISM

```
values (init, [sun, rain]) .  
values (tr(_), [sun, rain])
```

random variables and their values

```
:- set_sw (init, [0.5, 0.5]) .  
:- set_sw (tr(sun), [0.6, 0.4]) .  
:- set_sw (tr(rain), [0.2, 0.8]) .
```

probability distributions

```
weather (W, Time) :-  
    Time >= 0,  
    msw (init, W0) ,  
    w (0, Time, W0, W) .
```

set **W0** to random value of **init**

```
w (T, T, W, W) .  
w (Now, T, WNow, WT) :-  
    Now < T,  
    msw (tr(WNow), WNext) ,  
    Next is Now+1,  
    w (Next, T, WNext, WT) .
```

Weather in PRISM

```
values (init, [sun, rain]) .  
values (tr(_), [sun, rain])
```

random variables and their values

```
:- set_sw (init, [0.5, 0.5]) .  
:- set_sw (tr(sun), [0.6, 0.4]) .  
:- set_sw (tr(rain), [0.2, 0.8]) .
```

probability distributions

```
weather (W, Time) :-  
    Time >= 0,  
    msw (init, W0) ,  
    w (0, Time, W0, W) .
```

set **W0** to random value of **init**

```
w (T, T, W, W) .  
w (Now, T, WNow, WT) :-  
    Now < T,  
    msw (tr (WNow), WNext) ,  
    Next is Now+1,  
    w (Next, T, WNext, WT) .
```

set **WNext** to random
value of **tr (WNow)** , using
fresh value on every call

Weather in PRISM / ProbLog

```
values(init,[sun,rain]).
values(tr(_),[sun,rain]).

:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).
```

```
weather(W,Time) :-
    Time >= 0,
    msw(init,W0),
    w(0,Time,W0,W).
```

```
w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

```
0.5::init(sun) ;
0.5::init(rain) <- true.
0.6::tr(T,sun,sun) ;
0.4::tr(T,sun,rain) <- true.
0.2::tr(T,rain,sun) ;
0.8::tr(T,rain,rain) <- true.
```

```
weather(W,Time) :-
    Time >= 0,
    init(W0),
    w(0,Time,W0,W).
```

```
w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    tr(Now,WNow,WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

ProbLog needs to explicitly use
different facts at each call

Probabilistic Prologs: Two Views

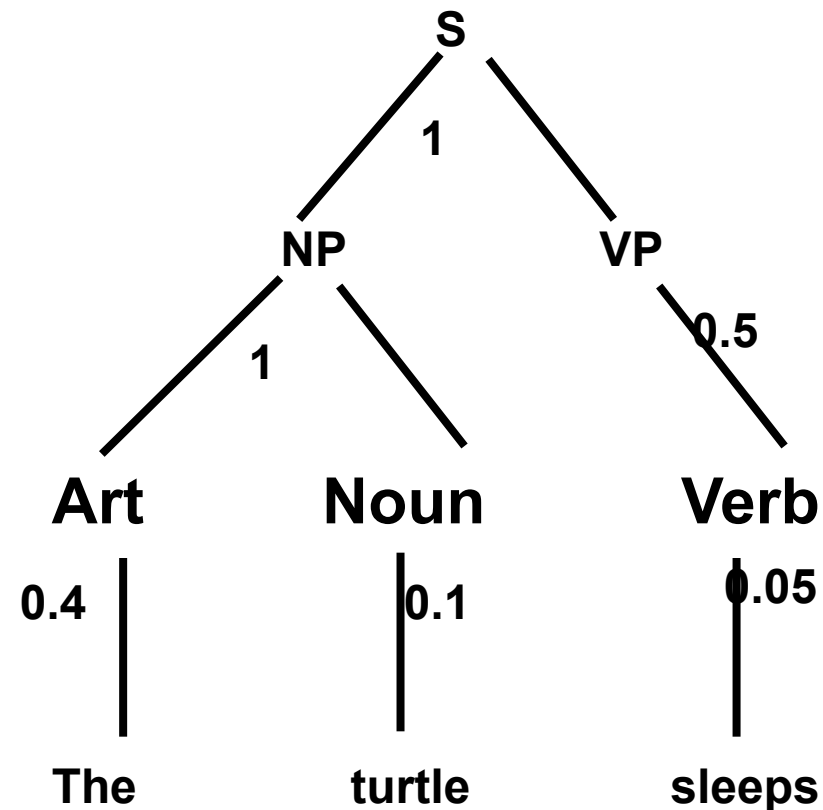
- Distribution semantics:
 - probability distribution over interpretations
 - degree of belief
- Stochastic Logic Programs (SLPs):
 - probability distribution over query answers
 - like in probabilistic grammars

Probabilistic Prologs: Two Views

- Distribution semantics:
 - probability distribution over interpretations
 - degree of belief
- Stochastic Logic Programs (SLPs):
 - probability distribution over query answers
 - like in probabilistic grammars

Probabilistic Context Free Grammars

1.0 : S → NP, VP
1.0 : NP → Art, Noun
0.6 : Art → a
0.4 : Art → the
0.1 : Noun → turtle
0.1 : Noun → turtles
...
0.5 : VP → Verb
0.5 : VP → Verb, NP
0.05 : Verb → sleep
0.05 : Verb → sleeps
....



$$P(\text{parse tree}) = 1 \times 1 \times 0.5 \times 1 \times 0.4 \times 0.05$$

PCFGs

$$P(\textit{parse tree}) = \prod_i p_i^{c_i}$$

where p_i is the probability of rule i
and c_i the number of times
it is used in the parse tree

$$P(\textit{sentence}) = \sum_{p:\textit{parsetree}} P(p)$$

Observe that derivations always succeed, that is
 $S \rightarrow T, Q$ and $T \rightarrow R, U$
always yields
 $S \rightarrow R, U, Q$

Probabilistic Definite Clause Grammar

1.0 $S \rightarrow NP(Num), VP(Num)$

1.0 $NP(Num) \rightarrow Art(Num), Noun(Num)$

0.6 $Art(sing) \rightarrow a$

0.2 $Art(sing) \rightarrow the$

0.2 $Art(plur) \rightarrow the$

0.1 $Noun(sing) \rightarrow turtle$

0.1 $Noun(plur) \rightarrow turtles$

...

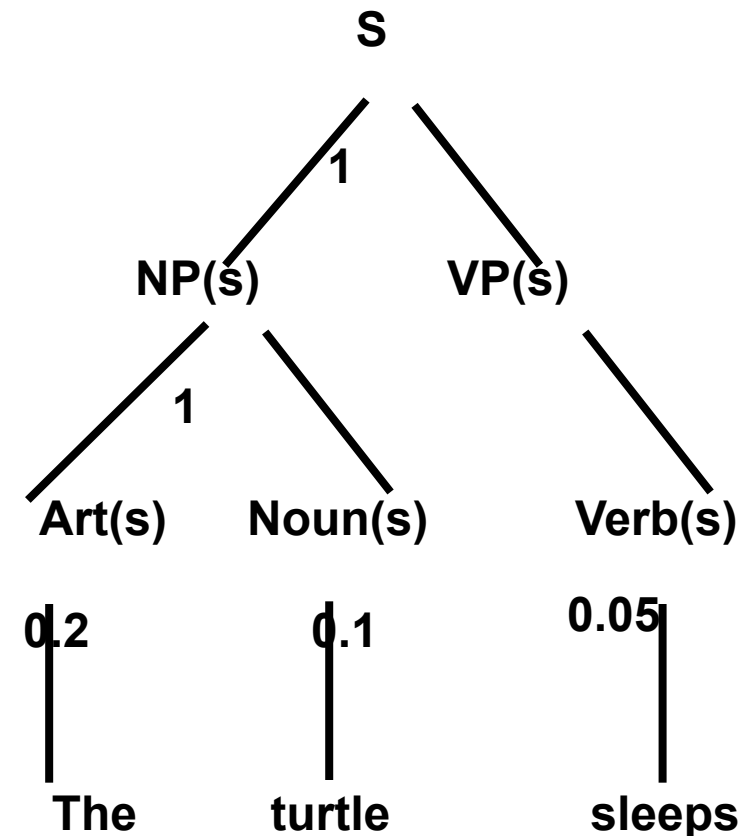
0.5 $VP(Num) \rightarrow Verb(Num)$

0.5 $VP(Num) \rightarrow Verb(Num), NP(Num)$

0.05 $Verb(sing) \rightarrow sleeps$

0.05 $Verb(plur) \rightarrow sleep$

....



$P(\text{derivation tree}) = 1 \times 1 \times 0.5 \times 0.1 \times 0.2 \times 0.05$

Stochastic Logic Programs

In SLP notation

1

```
sentence(A, B) :- noun_phrase(C, A, D), verb_phrase(C, D, B).
noun_phrase(A, B, C) :- article(A, B, D), noun(A, D, C).
verb_phrase(A, B, C) :- intransitive_verb(A, B, C).
```

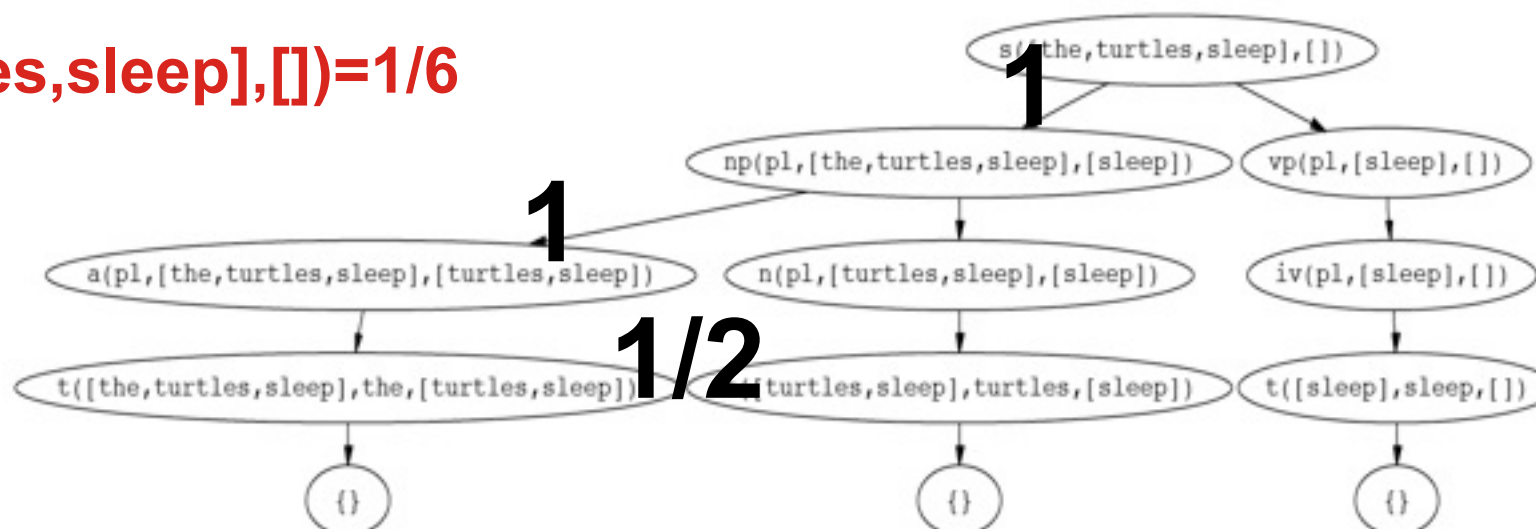
1/3

```
article(singular, A, B) :- terminal(A, a, B).
article(singular, A, B) :- terminal(A, the, B).
article(plural, A, B) :- terminal(A, the, B).
```

1/2

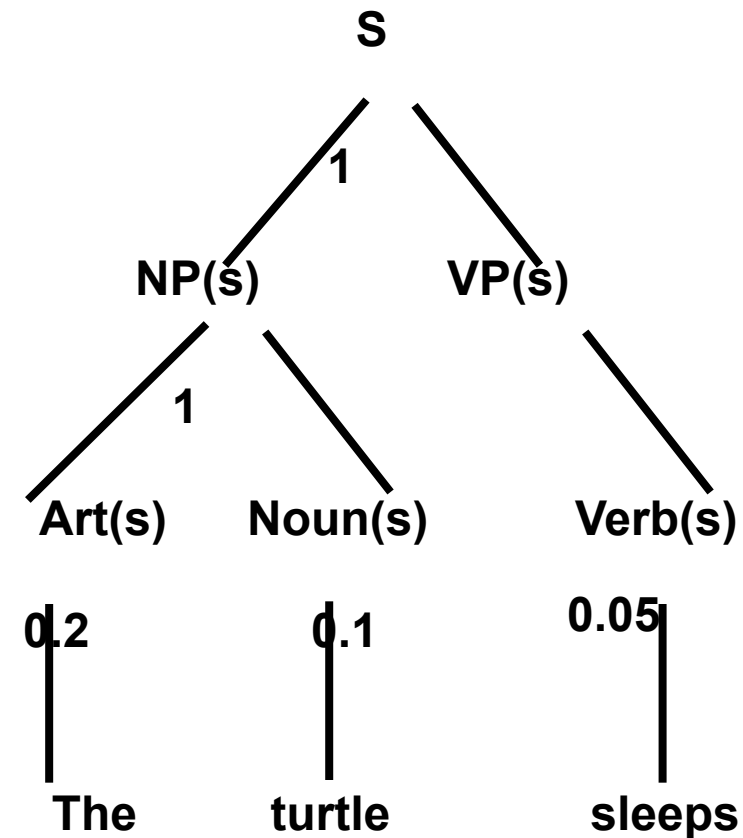
```
noun(singular, A, B) :- terminal(A, turtle, B).
noun(plural, A, B) :- terminal(A, turtles, B).
intransitive_verb(singular, A, B) :- terminal(A, sleeps, B).
intransitive_verb(plural, A, B) :- terminal(A, sleep, B).
terminal([A|B], A, B).
```

$P(s([the, turtles, sleep], [])) = 1/6$



Probabilistic DCG

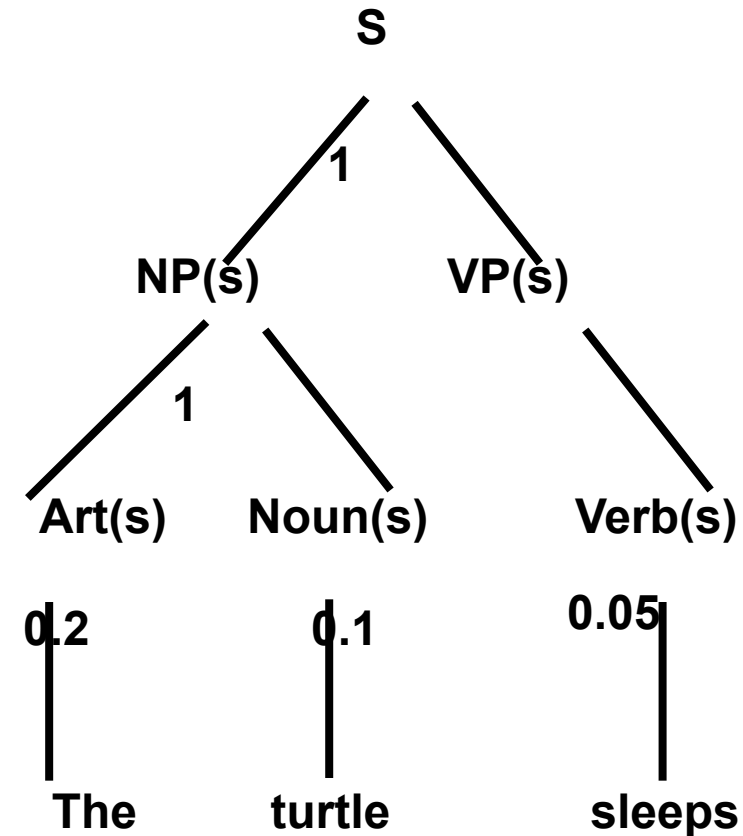
1.0 $S \rightarrow NP(Num), VP(Num)$
 1.0 $NP(Num) \rightarrow Art(Num), Noun(Num)$
 0.6 $Art(sing) \rightarrow a$
 0.2 $Art(sing) \rightarrow the$
 0.2 $Art(plur) \rightarrow the$
 0.1 $Noun(sing) \rightarrow turtle$
 0.1 $Noun(plur) \rightarrow turtles$
 ...
 0.5 $VP(Num) \rightarrow Verb(Num)$
 0.5 $VP(Num) \rightarrow Verb(Num), NP(Num)$
 0.05 $Verb(sing) \rightarrow sleeps$
 0.05 $Verb(plur) \rightarrow sleep$



P(derivation tree) = 1x1x.5x.1x .2 x.05

Probabilistic DCG

1.0 $S \rightarrow NP(Num), VP(Num)$
1.0 $NP(Num) \rightarrow Art(Num), Noun(Num)$
0.6 $Art(sing) \rightarrow a$
0.2 $Art(sing) \rightarrow the$
0.2 $Art(plur) \rightarrow the$
0.1 $Noun(sing) \rightarrow turtle$
0.1 $Noun(plur) \rightarrow turtles$
...
0.5 $VP(Num) \rightarrow Verb(Num)$
0.5 $VP(Num) \rightarrow Verb(Num), NP(Num)$
0.05 $Verb(sing) \rightarrow sleeps$
0.05 $Verb(plur) \rightarrow sleep$
....



$P(\text{derivation tree}) = 1 \times 1 \times .5 \times .1 \times .2 \times .05$

What about “A turtles sleeps” ?

SLPs

$$P_d(\textit{derivation}) = \prod_i p_i^{c_i}$$

where p_i is the probability of rule i
and c_i the number of times
it is used in the parse tree

Observe that some derivations now fail due to unification,
 $np(Num) \rightarrow art(Num), noun(Num)$ and $art(sing) \rightarrow a$
 $noun(plural) \rightarrow turtles$

Normalization necessary

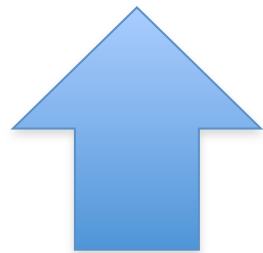
$$P_s(\textit{proof}) = \frac{P_d(\textit{proof})}{\sum_i P_d(\textit{proof}_i)}$$

ProPPR

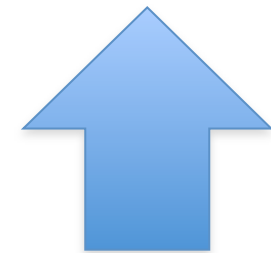
- A variation on SLPs
- Integrating concepts from Personalized Page Rank
- Fast inference and rule learning abilities
- Used by CMU group for NELL (Never Ending Learning)
- See [Wang et al., CIKM 13, [arXiv:1404.3301](https://arxiv.org/abs/1404.3301)]

Sample ProPPR program....

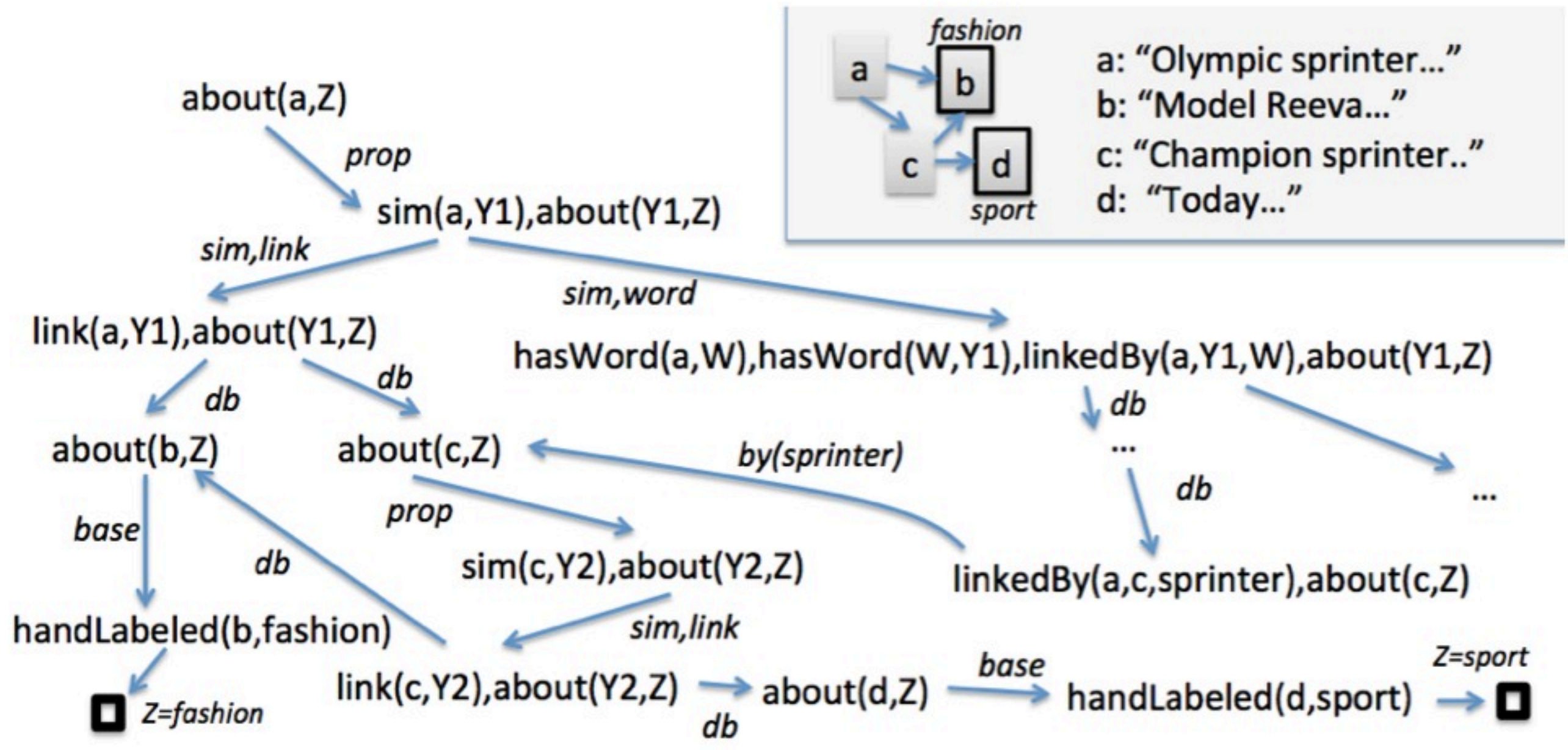
```
about(X,Z) :- handLabeled(X,Z)           # base.  
about(X,Z) :- sim(X,Y),about(Y,Z)       # prop.  
sim(X,Y) :- links(X,Y)                  # sim,link.  
sim(X,Y) :-  
    hasWord(X,W),hasWord(Y,W),  
    linkedBy(X,Y,W)                     # sim,word.  
linkedBy(X,Y,W) :- true                  # by(W).
```



Horn rules



features of rules
(vars from head ok)



.. and search space...

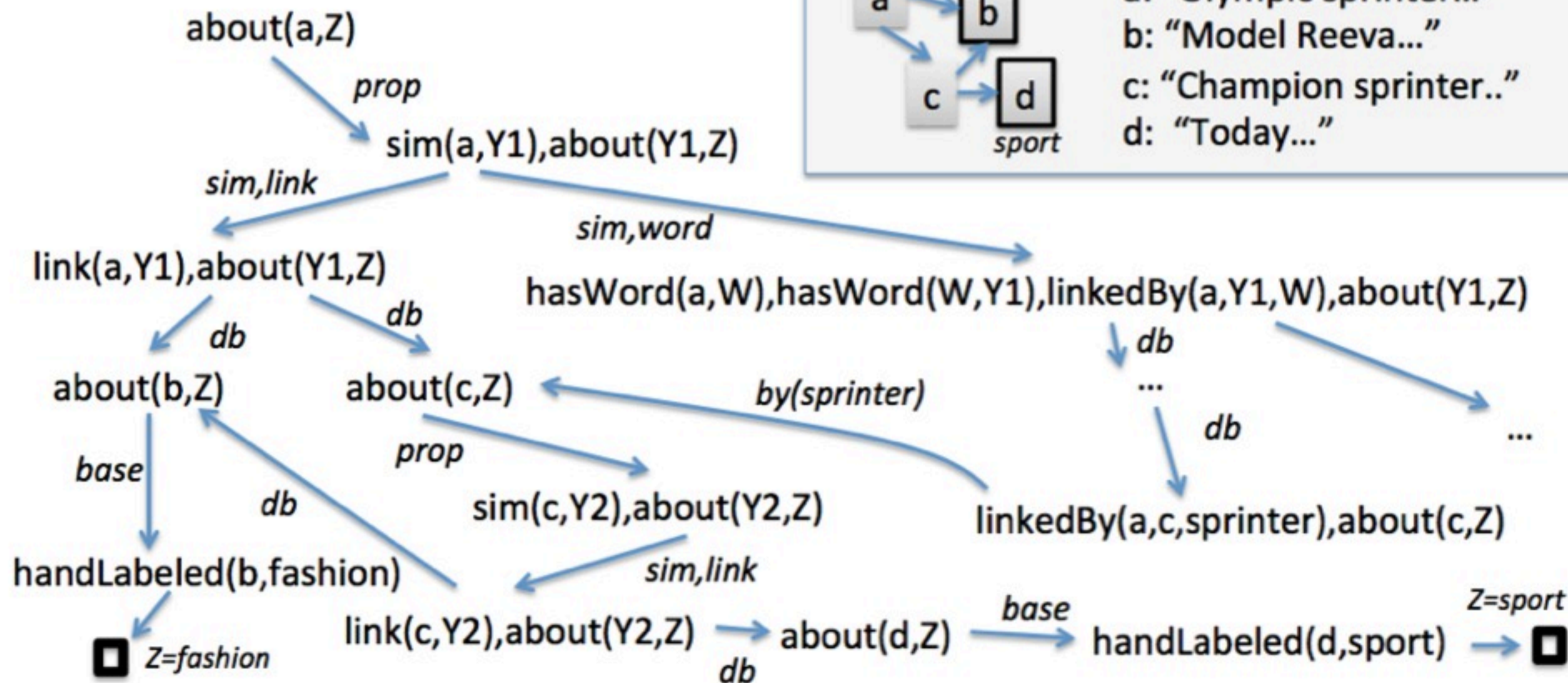
[Slide by William Cohen]

```

about(X,Z) :- handLabeled(X,Z)           # base.
about(X,Z) :- sim(X,Y),about(Y,Z)       # prop.
sim(X,Y) :- links(X,Y)                  # sim,link.
sim(X,Y) :-
    hasWord(X,W),hasWord(Y,W),
    linkedBy(X,Y,W)                     # sim,word.
linkedBy(X,Y,W) :- true                  # by(W).

```


D'oh! This is a graph!



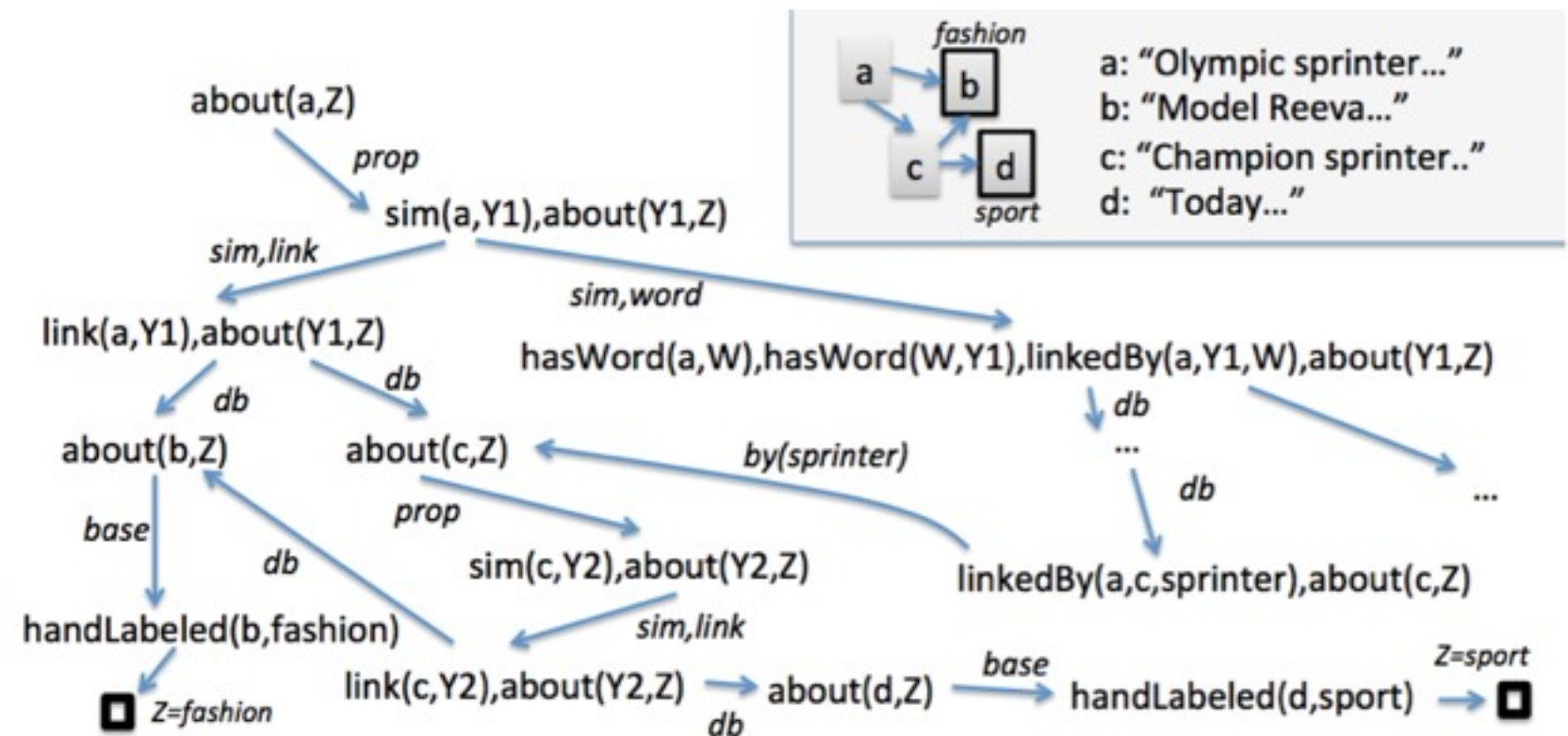
.. and search space...

```

about(X,Z) :- handLabeled(X,Z)           # base.
about(X,Z) :- sim(X,Y),about(Y,Z)        # prop.
sim(X,Y) :- links(X,Y)                   # sim,link.
sim(X,Y) :-
    hasWord(X,W),hasWord(Y,W),
    linkedBy(X,Y,W)                       # sim,word.
linkedBy(X,Y,W) :- true                   # by(W).

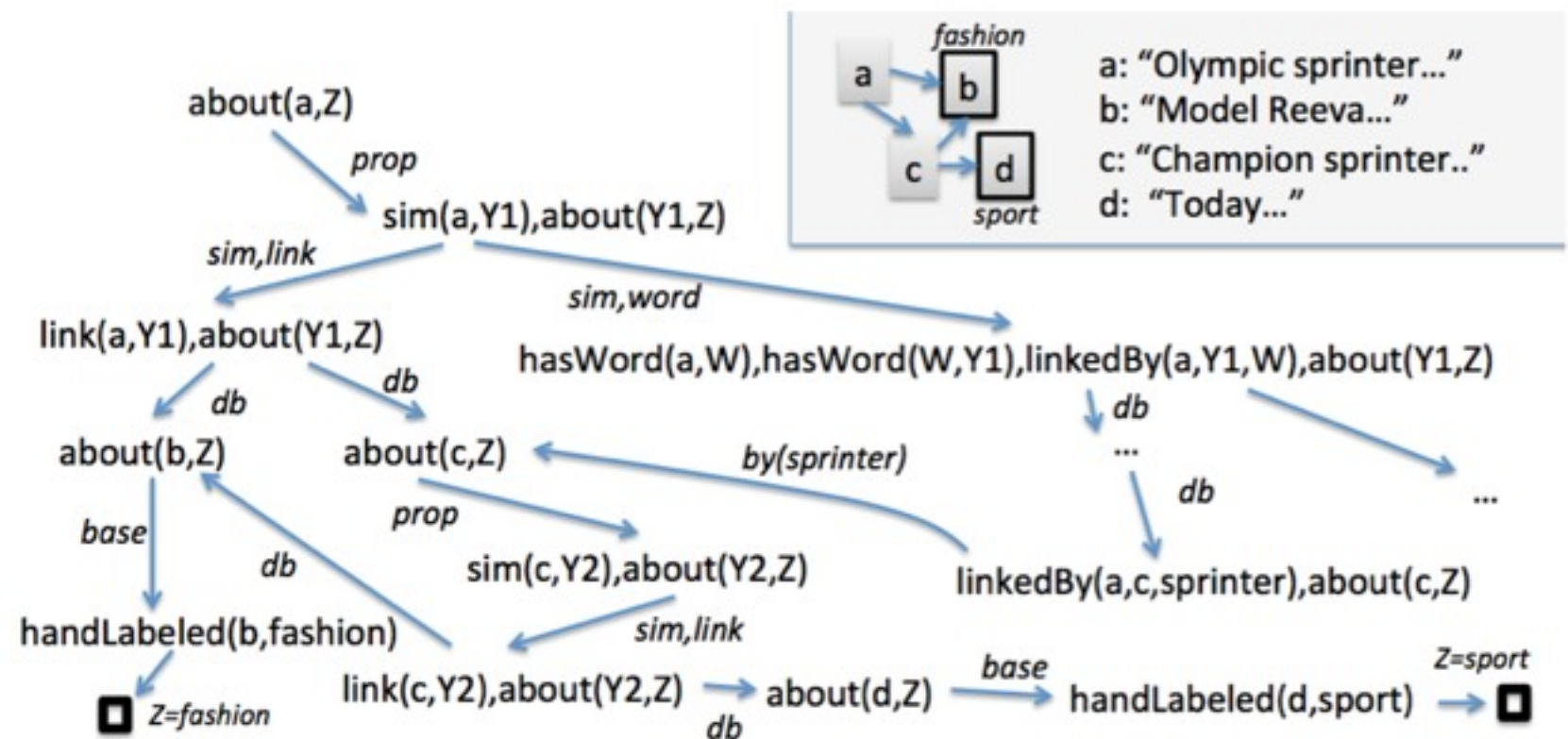
```

- Score for a query soln (e.g., “Z=sport” for “about(a,Z)”) depends on probability of reaching a \square node*
- learn transition probabilities based on **features** of the rules
- implicit “**reset**” transitions with ($p \geq \alpha$) back to query node
- Looking for answers supported by many short proofs



- Score for a query soln (e.g., “Z=sport” for “about(a,Z)”) depends on probability of reaching a \square node*
 - learn transition probabilities based on **features** of the rules
 - implicit “**reset**” transitions with ($p \geq \alpha$) back to query node
- Looking for answers supported by many short proofs

*Exactly as in Stochastic Logic Programs
[Cussens, 2001]

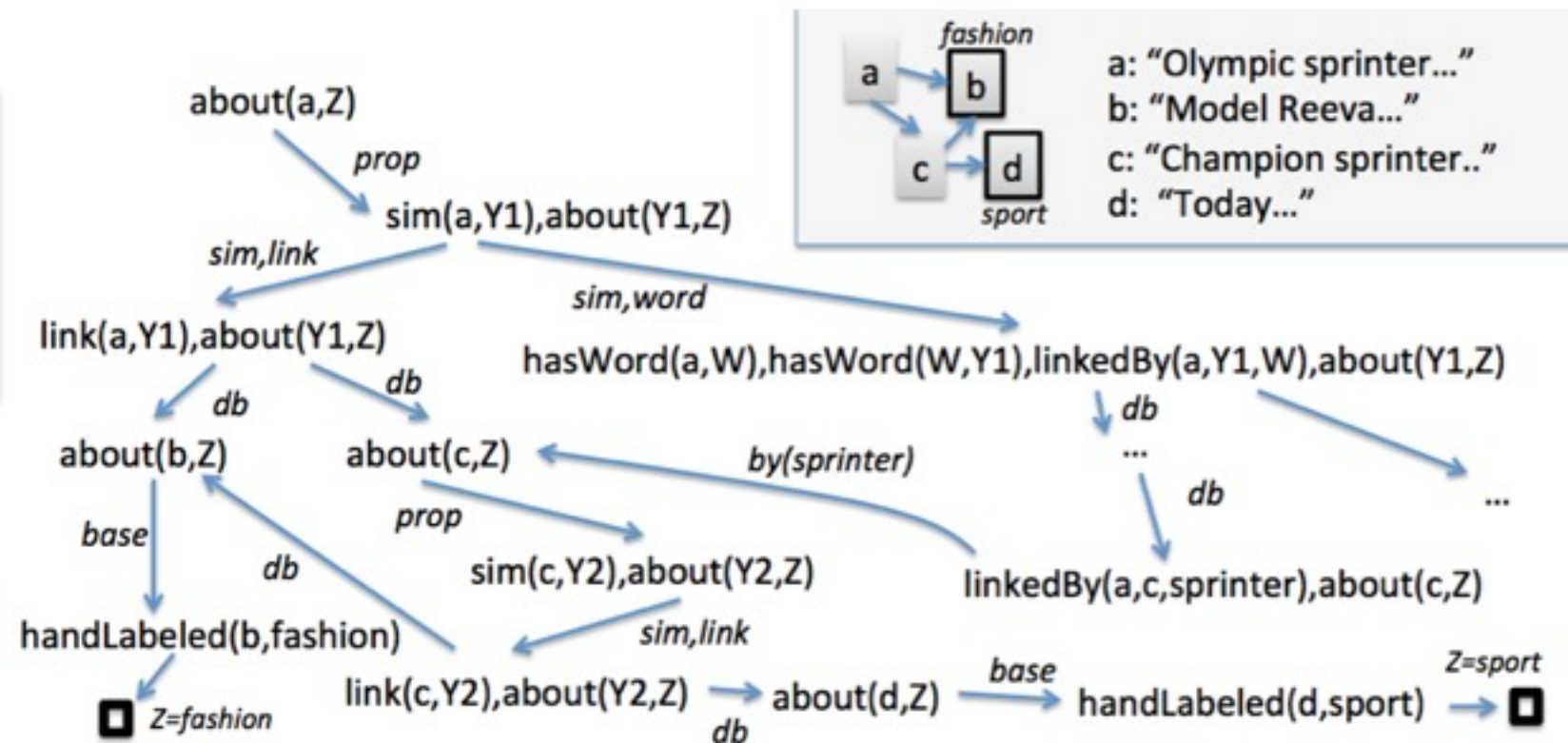


- Score for a query soln (e.g., “Z=sport” for “about(a,Z)”) depends on probability of reaching a \square node*
 - learn transition probabilities based on **features** of the rules
 - implicit “**reset**” transitions with ($p \geq \alpha$) back to query node
- Looking for answers supported by many short proofs

“Grounding” size is $O(1/\alpha\epsilon)$... ie independent of DB size \rightarrow fast approx incremental inference (Reid, Lang, Chung, 08)

*Exactly as in Stochastic Logic Programs [Cussens, 2001]

Learning: supervised variant of personalized PageRank (Backstrom & Leskovic, 2011)



Probabilistic Programming Languages outside LP

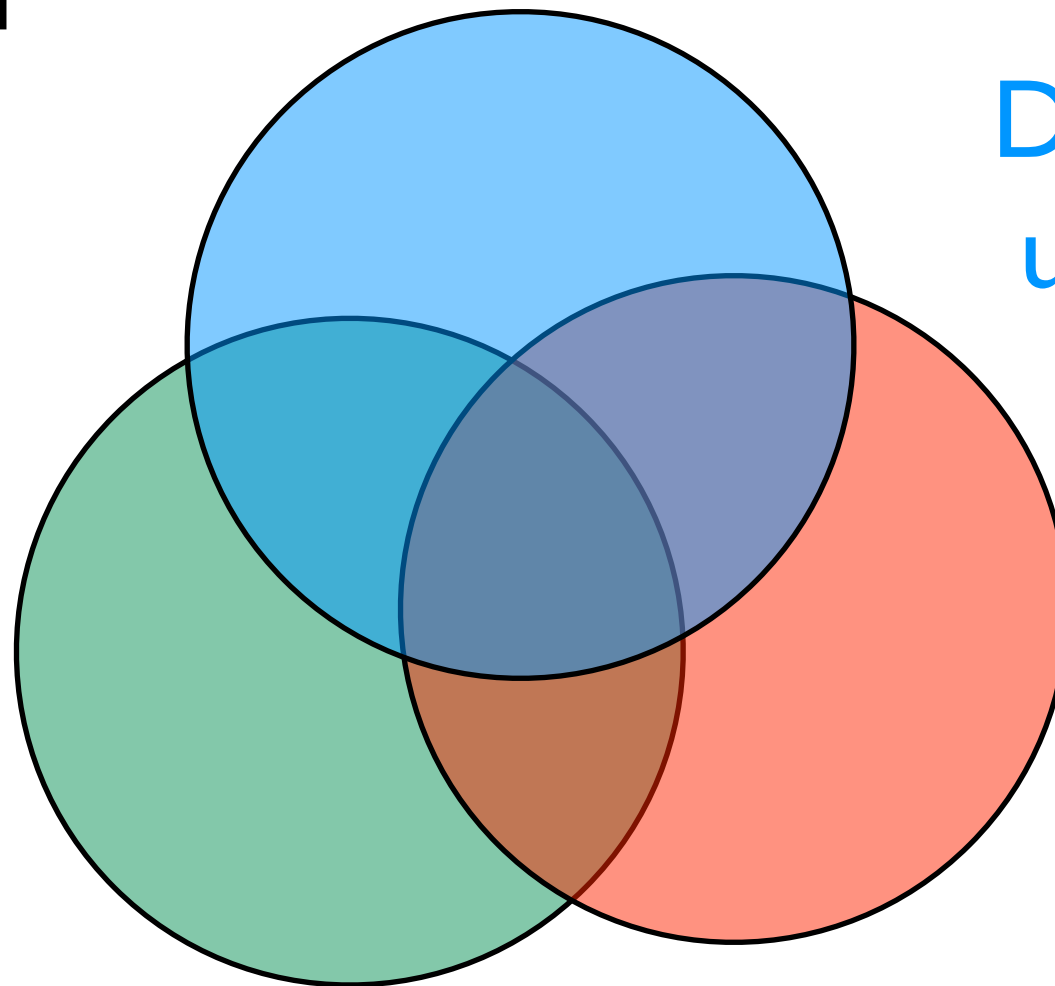
- IBAL [Pfeffer 01]
- Figaro [Pfeffer 09]
- Church [Goodman et al 08]
- BLOG [Milch et al 05]
- and many more appearing recently

Church

probabilistic functional
programming

[Goodman et al, UAI 08]

Reasoning with
relational data



Dealing with
uncertainty

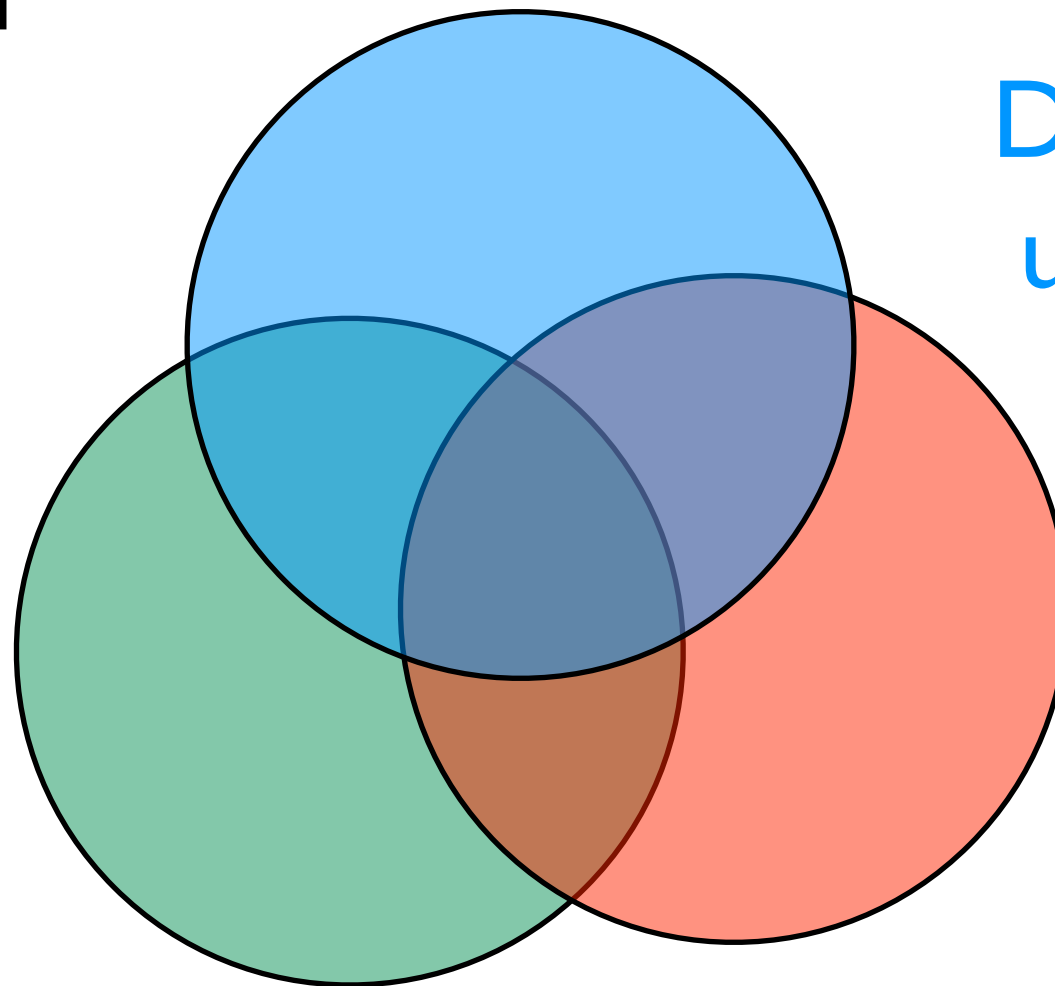
Learning

Church

probabilistic functional
programming

[Goodman et al, UAI 08]

functional
programming



Dealing with
uncertainty

Learning

```
(define plus5 (lambda (x) (+ x 5)))
```

```
(map plus5 '(1 2 3))
```

Church

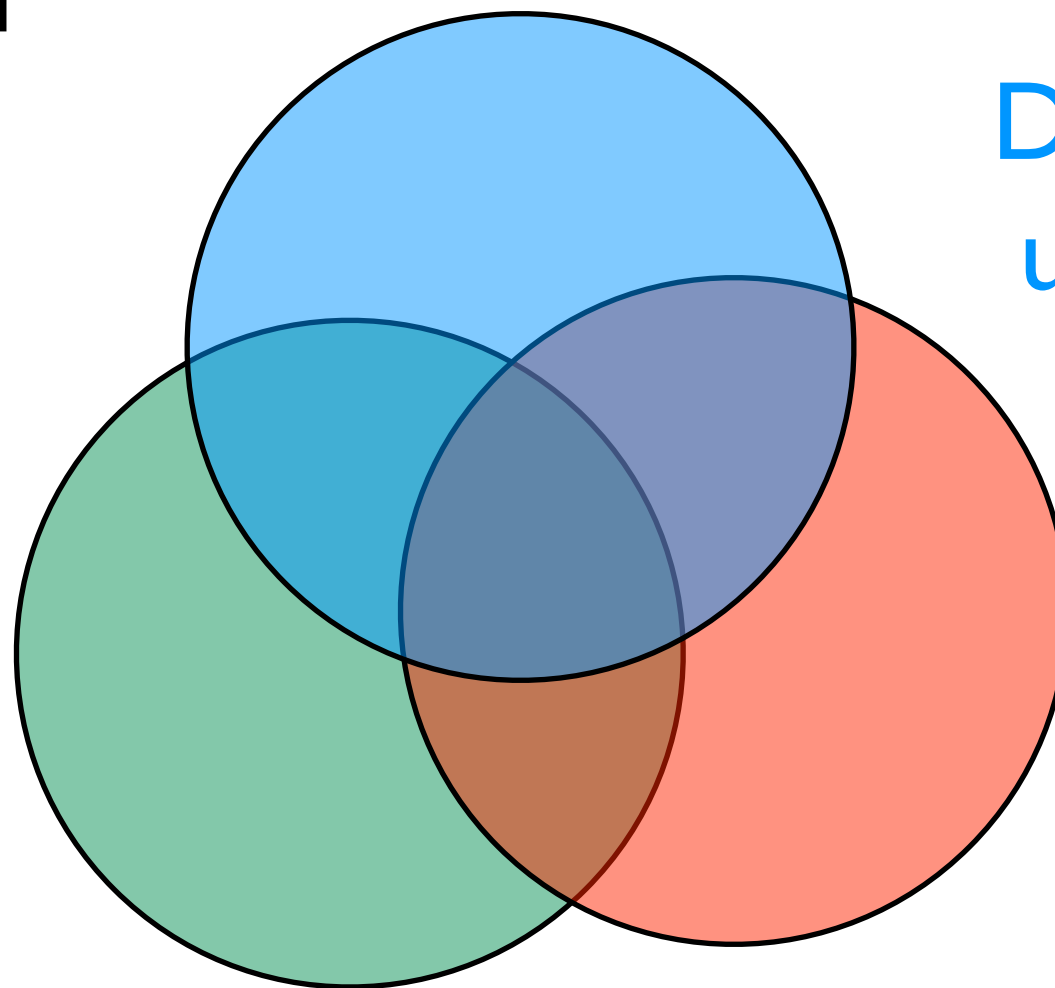
probabilistic functional
programming

[Goodman et al, UAI 08]

functional
programming

one execution

```
(define plus5 (lambda (x) (+ x 5)))  
  
(map plus5 '(1 2 3))
```



Dealing with
uncertainty

Learning

Church

probabilistic functional

programming

[Goodman et al, UAI 08]

```
(define randplus5  
  (lambda (x) (if (flip 0.6)  
                  (+ x 5)  
                  x)))
```

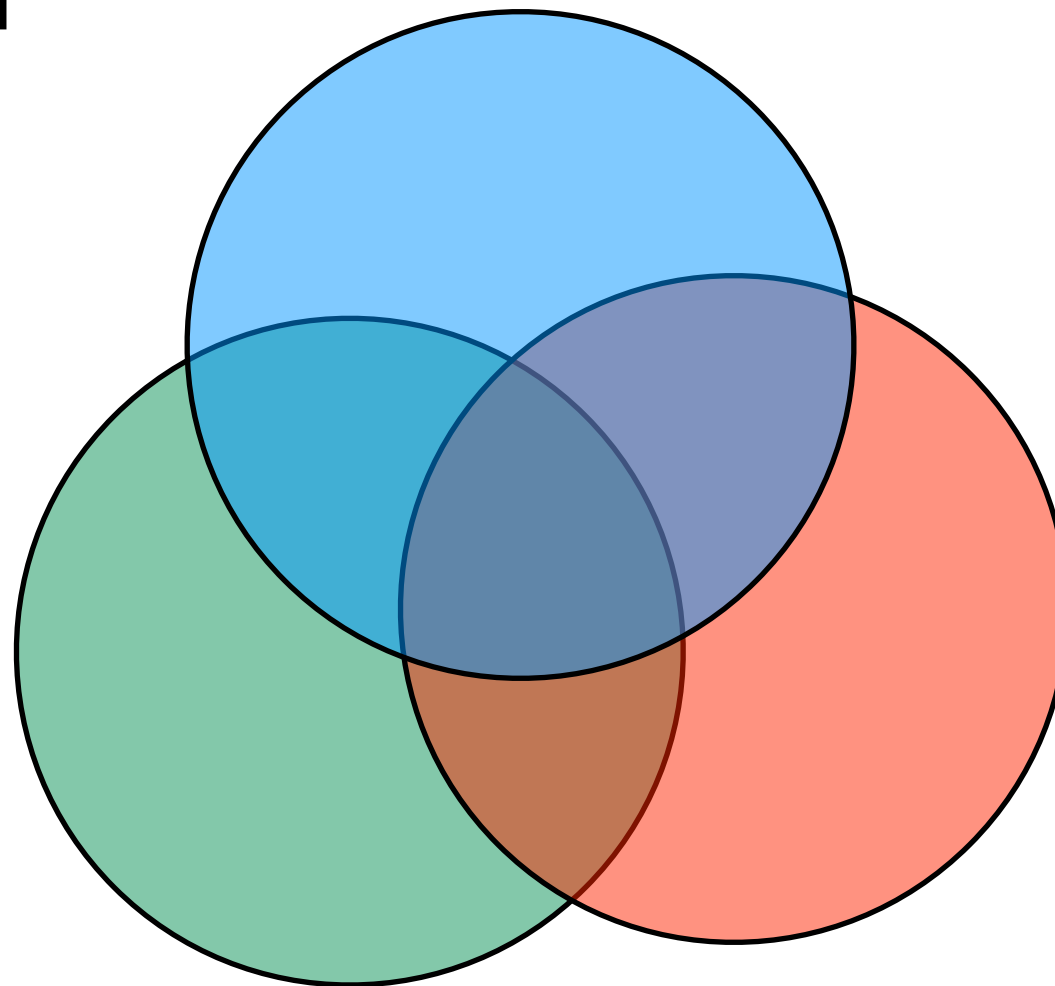
```
(map randplus5 '(1 2 3))
```

random
primitives

functional
programming

one execution

```
(define plus5 (lambda (x) (+ x 5)))  
  
(map plus5 '(1 2 3))
```



Learning

Church

probabilistic functional
programming

[Goodman et al, UAI 08]

**several
possible
executions**

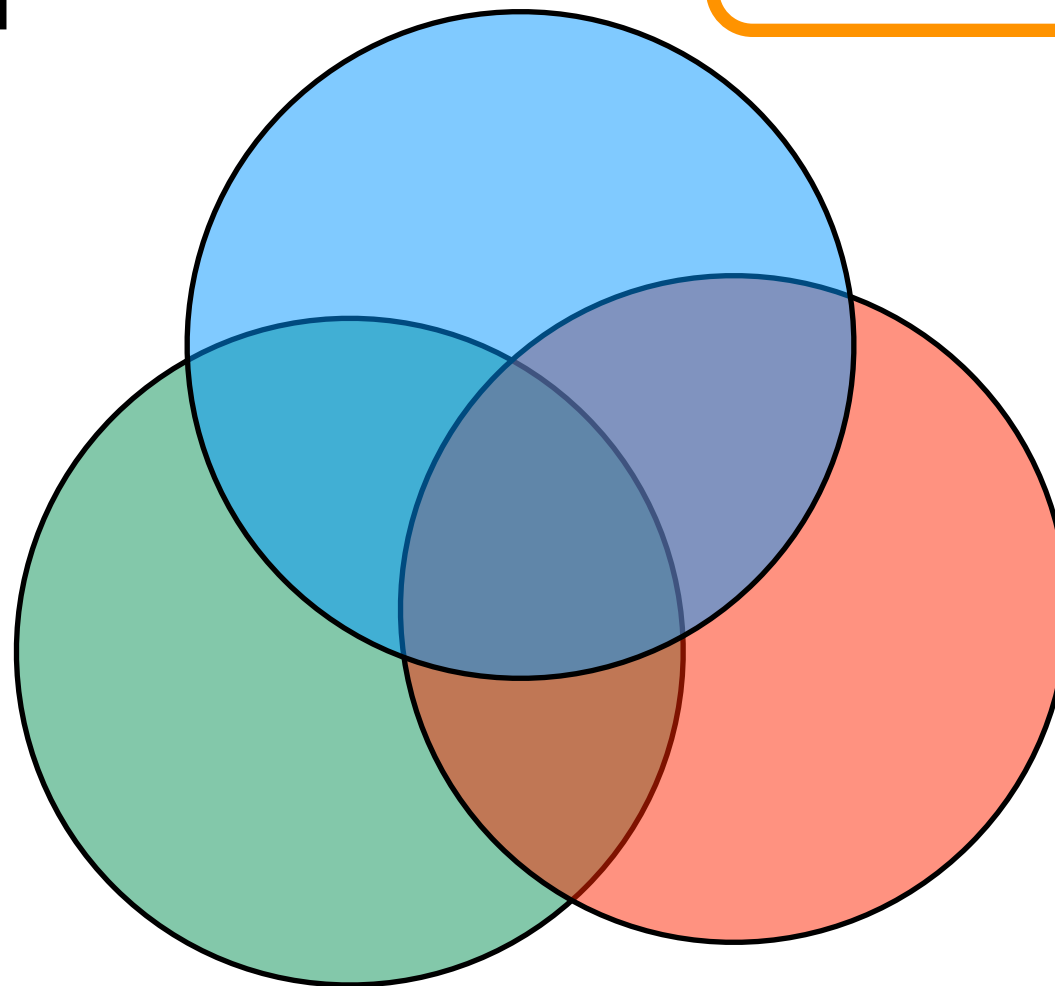
```
(define randplus5  
  (lambda (x) (if (flip 0.6)  
                  (+ x 5)  
                  x)))  
  
(map randplus5 '(1 2 3))
```

random
primitives

functional
programming

one execution

```
(define plus5 (lambda (x) (+ x 5)))  
  
(map plus5 '(1 2 3))
```



Learning

Church

probabilistic functional
programming

[Goodman et al, UAI 08]

**several
possible
executions**

```
(define randplus5  
  (lambda (x) (if (flip 0.6)  
                  (+ x 5)  
                  x)))  
  
(map randplus5 '(1 2 3))
```

random
primitives

probabilistic primitives + functional program
→ distribution over possible executions

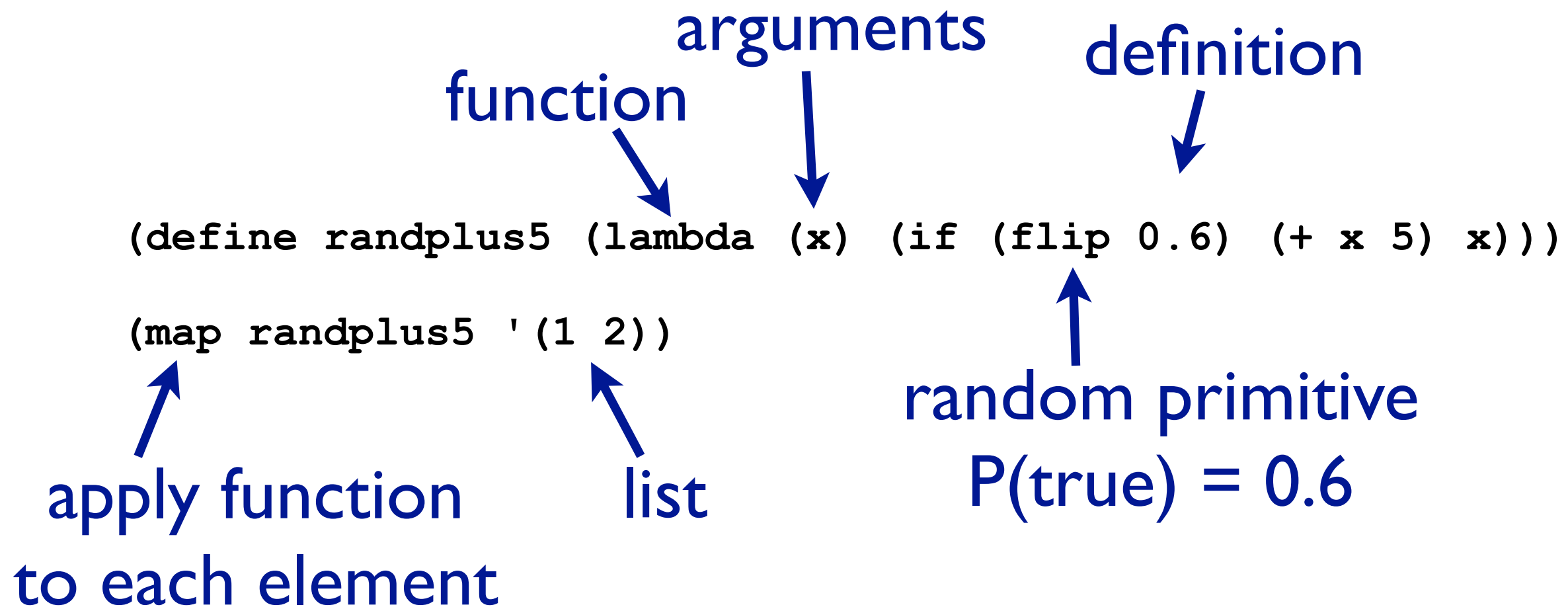
functional
programming

Learning

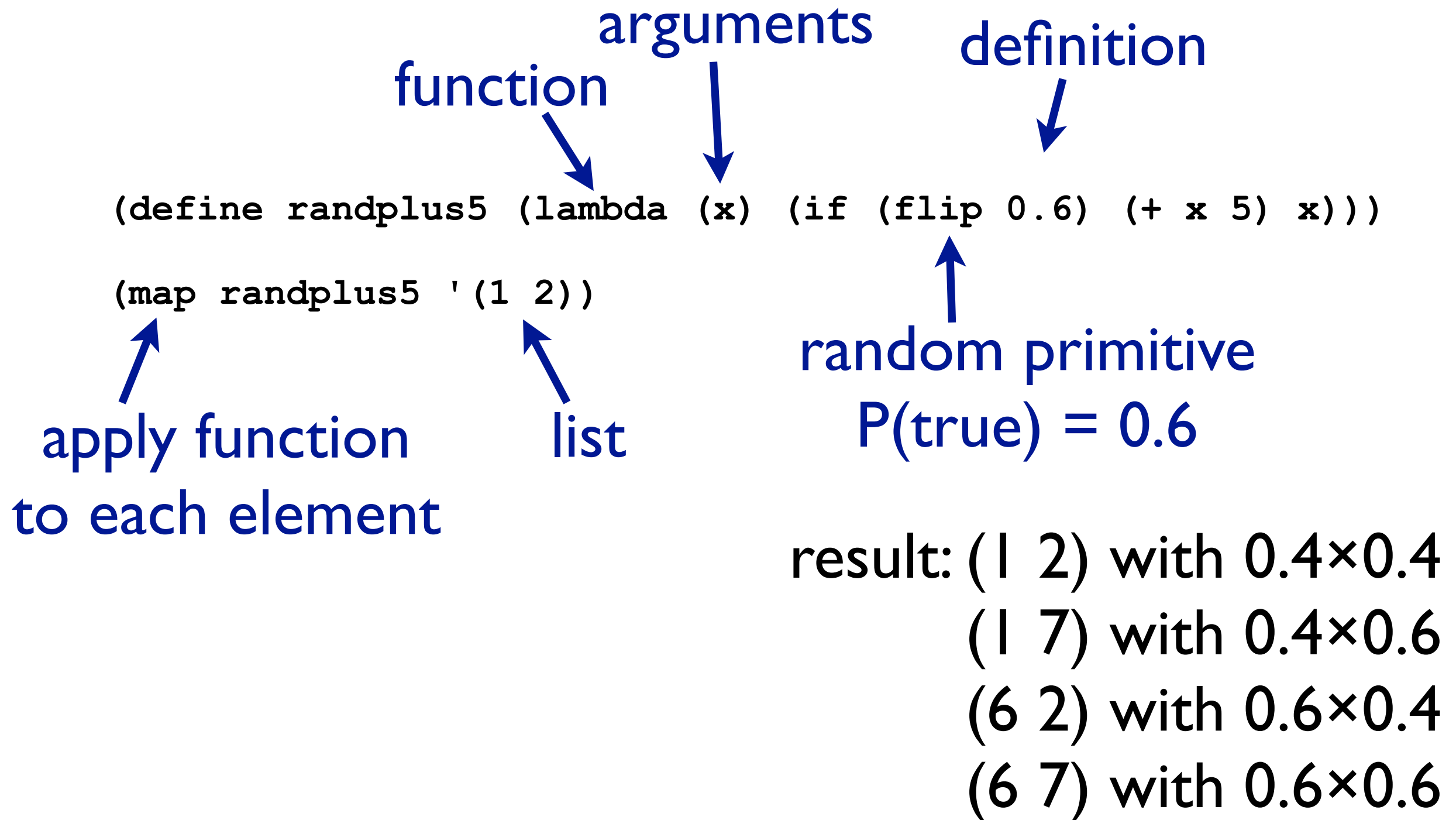
one execution

```
(define plus5 (lambda (x) (+ x 5)))  
  
(map plus5 '(1 2 3))
```


Church Example



Church Example



in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))  
  
(map randplus5 '(1 2))
```

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))
```

```
(map randplus5 '(1 2))
```

```
0.4::p5(N,N) ; 0.6::p5(N,M) <- M is N+5.
```

```
lp5([], []).
```

```
lp5([N|L], [M|K]) :-
```

```
    p5(N,M),
```

```
    lp5(L,K).
```

```
query(lp5([1,2],_)).
```

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))
```

```
(map randplus5 '(1 2))
```

```
0.4::p5(N,N) ; 0.6::p5(N,M) <- M is N+5.
```

```
lp5([], []).
```

```
lp5([N|L], [M|K]) :-
```

```
    p5(N,M),
```

```
    lp5(L,K).
```

```
query(lp5([1,2],_)).
```

result: (1 2) with 0.4×0.4
(1 7) with 0.4×0.6
(6 2) with 0.6×0.4
(6 7) with 0.6×0.6

results for [1,1]?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))
```

```
(map randplus5 '(1 1))
```

```
0.4 :: p5(N,N) ; 0.6 :: p5(N,M) <- M is N+5.
```

```
lp5([], []).
```

```
lp5([N|L], [M|K]) :-
```

```
    p5(N,M),
```

```
    lp5(L,K).
```

```
query(lp5([1,1],_)).
```

results for [1,1]?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))
```

```
(map randplus5 '(1 1))
```

Church result: (1 1) with 0.4×0.4

(1 6) with 0.4×0.6

(6 1) with 0.6×0.4

(6 6) with 0.6×0.6

```
0.4 :: p5 (N,N) ; 0.6 :: p5 (N,M) <- M is N+5.
```

```
lp5([], []).
```

```
lp5([N|L], [M|K]) :-
```

```
  p5(N,M),
```

```
  lp5(L,K).
```

```
query(lp5([1,1],_)).
```

results for [1,1]?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))
```

```
(map randplus5 '(1 1))
```

Church result: (1 1) with 0.4×0.4

(1 6) with 0.4×0.6

(6 1) with 0.6×0.4

(6 6) with 0.6×0.6

```
0.4::p5(N,N);0.6::p5(N,M) <- M is N+5.
```

```
lp5([],[]).
```

```
lp5([N|L],[M|K]) :-
```

```
  p5(N,M),
```

```
  lp5(L,K).
```

```
query(lp5([1,1],_)).
```

ProbLog result: (1 1) with 0.4

(1 6) with 0.0

(6 1) with 0.0

(6 6) with 0.6

results for [1,1]?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))
```

```
(map randplus5 '(1 1))
```

Church result: (1 1) with 0.4×0.4

(1 6) with 0.4×0.6

(6 1) with 0.6×0.4

(6 6) with 0.6×0.6

```
0.4 :: p5 (N,N) ; 0.6 :: p5 (N,M) <- M is N+5.
```

```
lp5 ([], []) .
```

```
lp5 ([N|L], [M|K]) :-
```

```
  p5 (N,M) ,
```

```
  lp5 (L,K) .
```

```
query (lp5 ([1,1], _)) .
```

ProbLog result: (1 1) with 0.4

(1 6) with 0.0

(6 1) with 0.0

(6 6) with 0.6

stochastic memoization

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))  
  
(map randplus5 '(1 1))
```

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))
```

```
(map randplus5 '(1 1))
```

```
0.4::p5(N,N,ID);0.6::p5(N,M,ID) <- M is N+5.
```

```
lp5([],[]).
```

```
lp5([N|L],[M|K]) :-
```

```
    p5(N,M,L),
```

```
    lp5(L,K).
```

```
query(lp5([1,1],_)).
```

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))  
  
(map randplus5 '(1 1))
```

```
0.4::p5(N,N,ID);0.6::p5(N,M,ID) <- M is N+5.  
lp5([],[]).  
lp5([N|L],[M|K]) :-  
    p5(N,M,L),  
    lp5(L,K).
```

identifier distinguishes calls

```
query(lp5([1,1],_)).
```

Stochastic Memoization

```
(define randplus5 (mem (lambda (x) (if (flip 0.6) (+ x 5) x))))  
(map randplus5 '(1 1))
```



remember first value &
reuse for all later calls

Stochastic Memoization

```
(define randplus5 (mem (lambda (x) (if (flip 0.6) (+ x 5) x))))  
(map randplus5 '(1 1))
```



remember first value &
reuse for all later calls

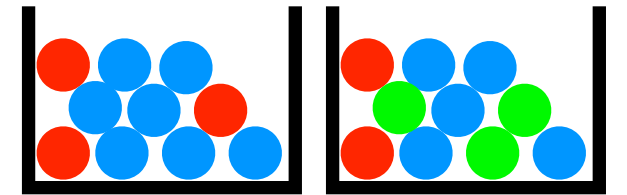
ProbLog always memoizes

PRISM never memoizes

Church allows fine-grained choice

Church by example:

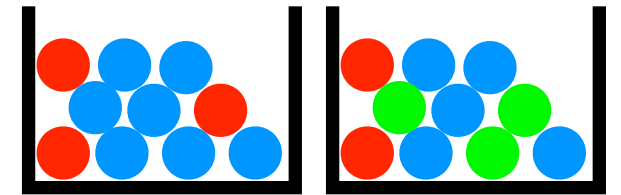
A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

Church by example:

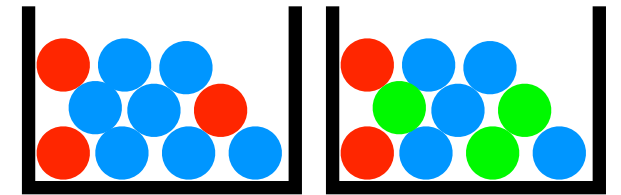
A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

Church by example:

A bit of gambling

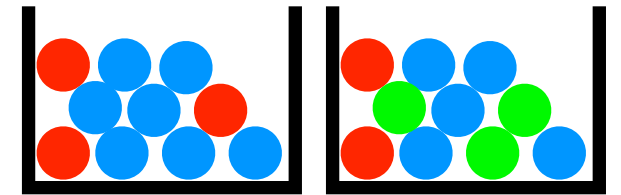


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

Church by example:

A bit of gambling

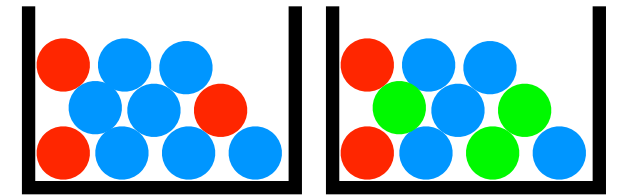


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

Church by example:

A bit of gambling



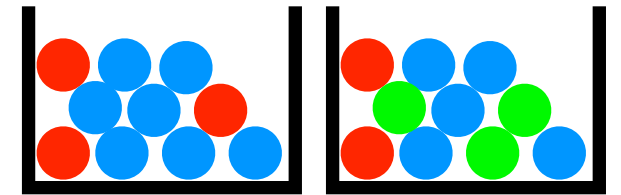
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

```
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))
```

Church by example:

A bit of gambling

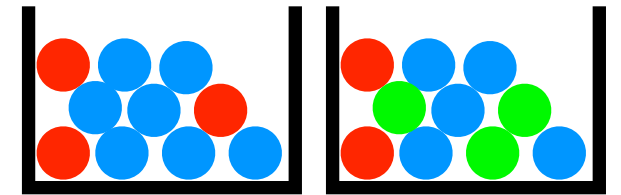


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5))))))
```

Church by example:

A bit of gambling

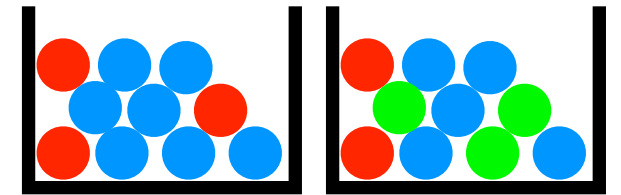


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5))))))
```

Church by example:

A bit of gambling

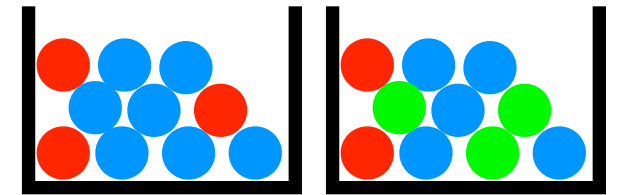


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))
```

Church by example:

A bit of gambling

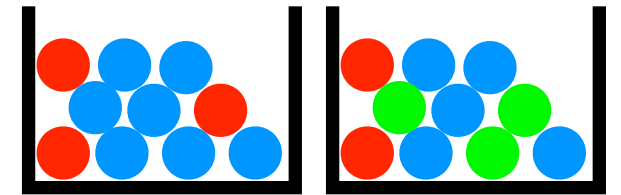


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5))))))  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
(define win1 (and (heads) redball))
```

Church by example:

A bit of gambling

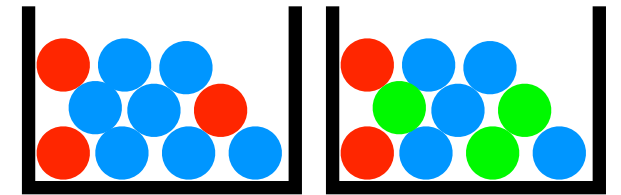


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5))))))  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
(define win1 (and (heads) redball))
```


Church by example:

A bit of gambling

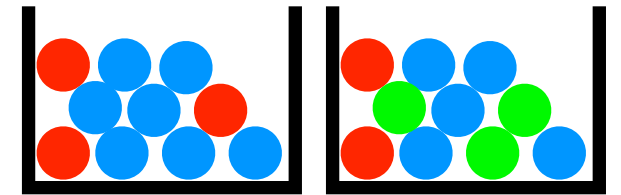


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5))))))  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
(define win1 (and (heads) redball))  
(define win2 (equal? (color1) (color2)))
```

Church by example:

A bit of gambling

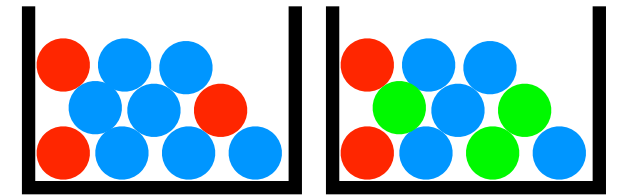


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5))))))  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
(define win1 (and (heads) redball))  
(define win2 (equal? (color1) (color2)))
```

Church by example:

A bit of gambling

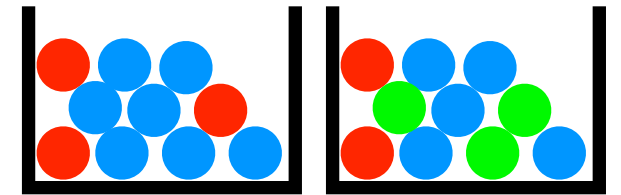


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5))))))  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
(define win1 (and (heads) redball))  
(define win2 (equal? (color1) (color2)))  
(define win (or win1 win2))
```

Church by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))  
  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))  
  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
  
(define win1 (and (heads) redball))  
  
(define win2 (equal? (color1) (color2)))  
  
(define win (or win1 win2))
```

Sampling execution

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
(define win1 (and (heads) redball))  
(define win2 (equal? (color1) (color2)))  
(define win (or win1 win2))
```

win ← query

Sampling execution

```
(define heads (mem (lambda () (flip 0.4))))  
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
(define color2 (mem (lambda ()  
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))  
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
(define win1 (and (heads) redball))  
(define win2 (equal? (color1) (color2)))  
(define win (or win1 win2))
```

win ← query

#t

Marginals via enumeration

(enumeration-query

```
(define heads (mem (lambda () (flip 0.4))))
```

```
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue)))))
```

```
(define color2 (mem (lambda ()  
                    (multinomial '(red green blue) '(0.2 0.3 0.5)))))
```

```
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))
```

```
(define win1 (and (heads) redball))
```

```
(define win2 (equal? (color1) (color2)))
```

```
(define win (or win1 win2))
```

win ← query

true)
← evidence

Marginals via enumeration

(enumeration-query

```
(define heads (mem (lambda () (flip 0.4))))
```

```
(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue)))))
```

```
(define color2 (mem (lambda ()  
                    (multinomial '(red green blue) '(0.2 0.3 0.5)))))
```

```
(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))
```

```
(define win1 (and (heads) redball))
```

```
(define win2 (equal? (color1) (color2)))
```

```
(define win (or win1 win2))
```

win ← query

true)
← evidence

```
((#f #t) (0.438 0.562))
```


Histogram via sampling

```
(repeat 1000 (lambda ()  
  (rejection-query  
  
    (define heads (mem (lambda () (flip 0.4))))  
  
    (define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))  
  
    (define color2 (mem (lambda ()  
      (multinomial '(red green blue) '(0.2 0.3 0.5)))))  
  
    (define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))  
  
    (define win1 (and (heads) redball))  
  
    (define win2 (equal? (color1) (color2)))  
  
    (define win (or win1 win2))
```

win ← query

true)))
← evidence

Histogram via sampling

```
(repeat 1000 (lambda ()
  (rejection-query

    (define heads (mem (lambda () (flip 0.4))))

    (define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

    (define color2 (mem (lambda ()
      (multinomial '(red green blue) '(0.2 0.3 0.5)))))

    (define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

    (define win1 (and (heads) redball))

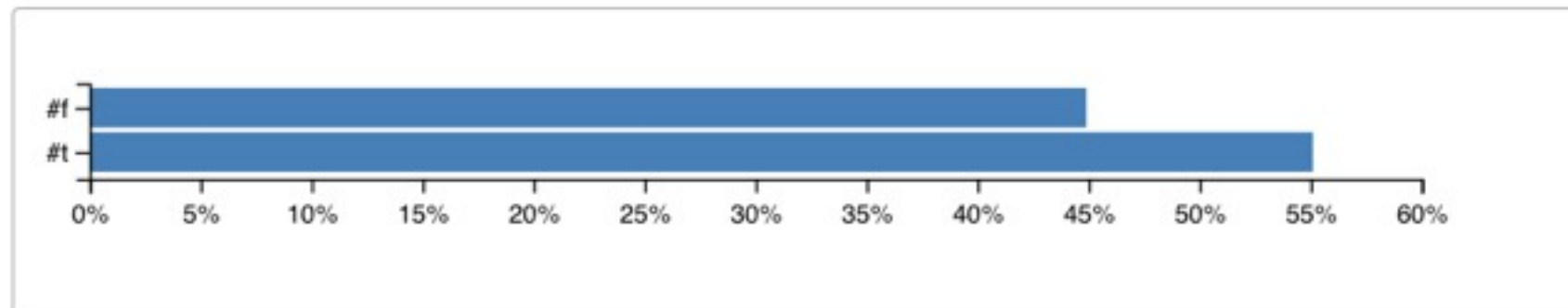
    (define win2 (equal? (color1) (color2)))

    (define win (or win1 win2))

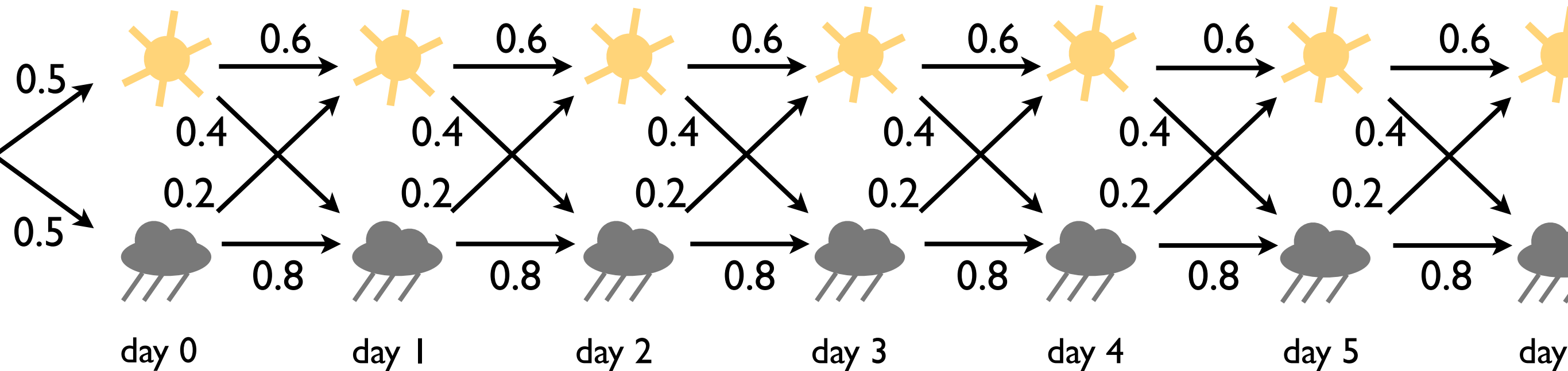
    win
  true ))))
```

win ← query

← evidence



Church by example: Rain or sun?

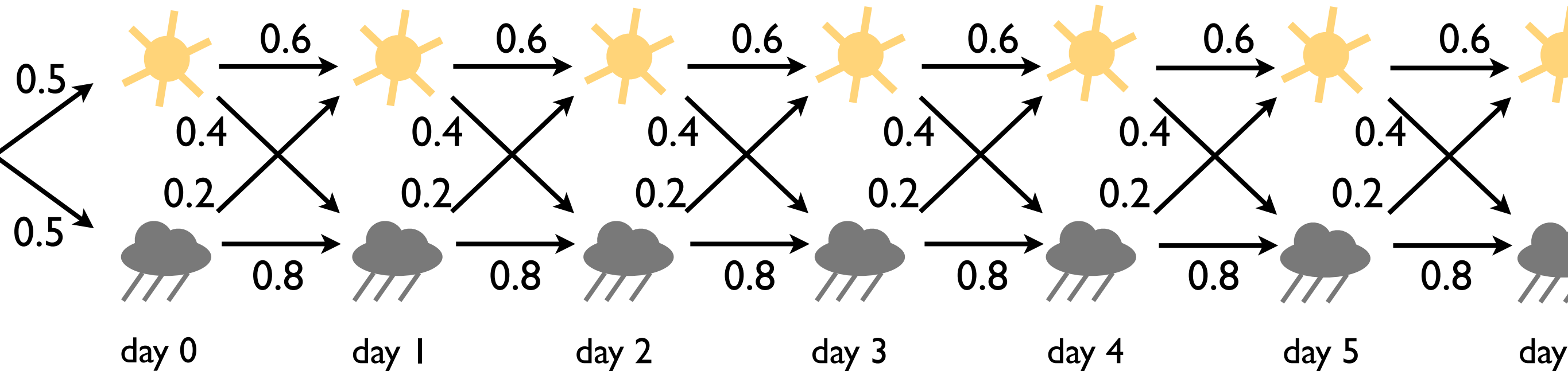


```
(define weather (mem (lambda (day) (if (equal? day 0)
                                         (weather0)
                                         (weatherN day (- day 1))))))

(define weather0 (lambda () (if (flip 0.5) 'sun 'rain)))

(define weatherN (lambda (today yesterday)
  (if (equal? (weather yesterday) 'rain)
      (if (flip 0.2) 'sun 'rain)
      (if (flip 0.6) 'sun 'rain))))
```

Church by example: Rain or sun?



```
(define weather (mem (lambda (day) (if (equal? day 0)
                                         (weather0)
                                         (weatherN day (- day 1))))))

(define weather0 (lambda () (if (flip 0.5) 'sun 'rain)))

(define weatherN (lambda (today yesterday)
  (if (equal? (weather yesterday) 'rain)
      (if (flip 0.2) 'sun 'rain)
      (if (flip 0.6) 'sun 'rain))))

(list (weather 0) (weather 1) (weather 2))
```

Church Summary

- functional programming + random primitives
- probabilistic generative model
- stochastic memoization
- sampling
- increasing number of probabilistic programming languages using various underlying paradigms

ProbLog	PRISM	Church
probabilistic facts & choices	probabilistic choices	random primitives
all RVs memoized	no RVs memoized	user-defined per RV
Prolog	Prolog with mutually exclusive derivations	λ -calculus functions
distribution over worlds	distribution over derivations / answers	distribution over computations / answers

Roadmap

- Modeling
- Reasoning
- Language extensions
- Advanced topics

... with some detours on the way

Reasoning

- Exact inference with knowledge compilation
 - using proofs
 - using models
 - under additional assumptions
- Approximate inference

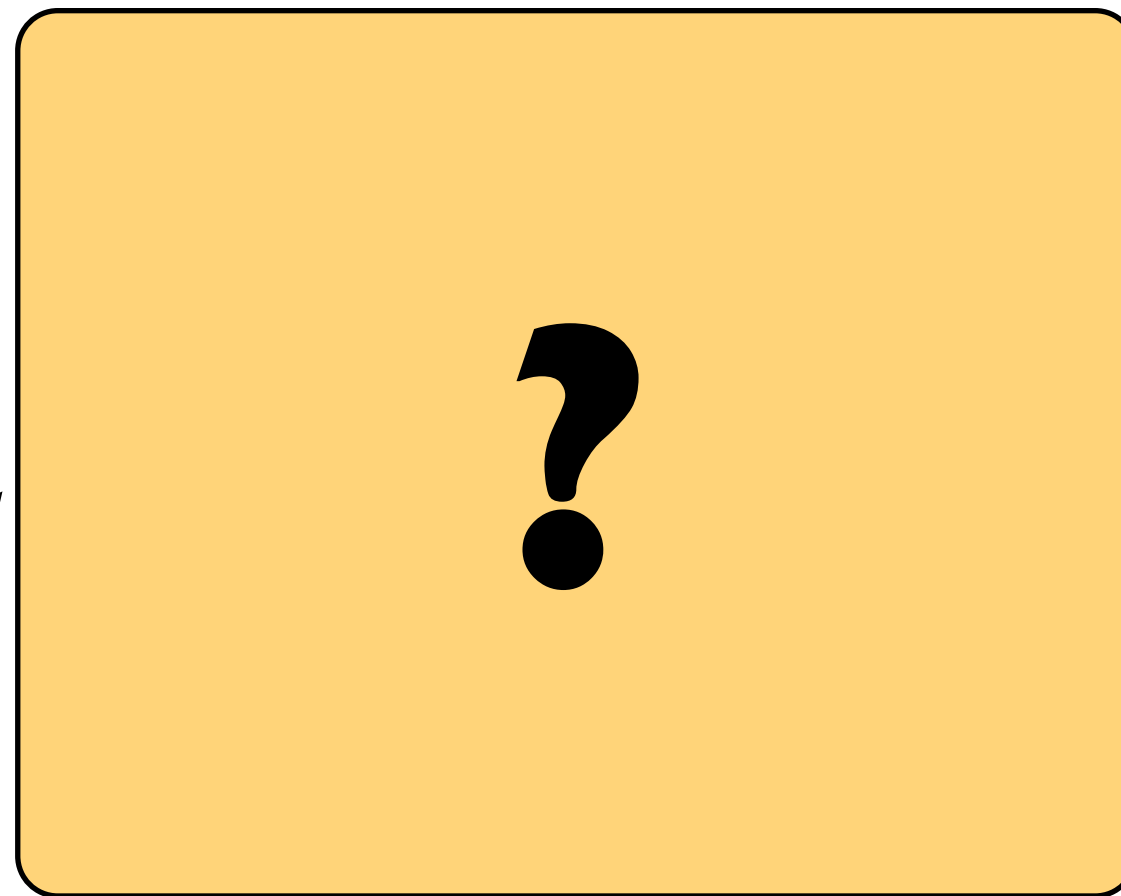
Answering Questions

Given:

program

queries

evidence



Find:

marginal
probabilities

conditional
probabilities

MPE state

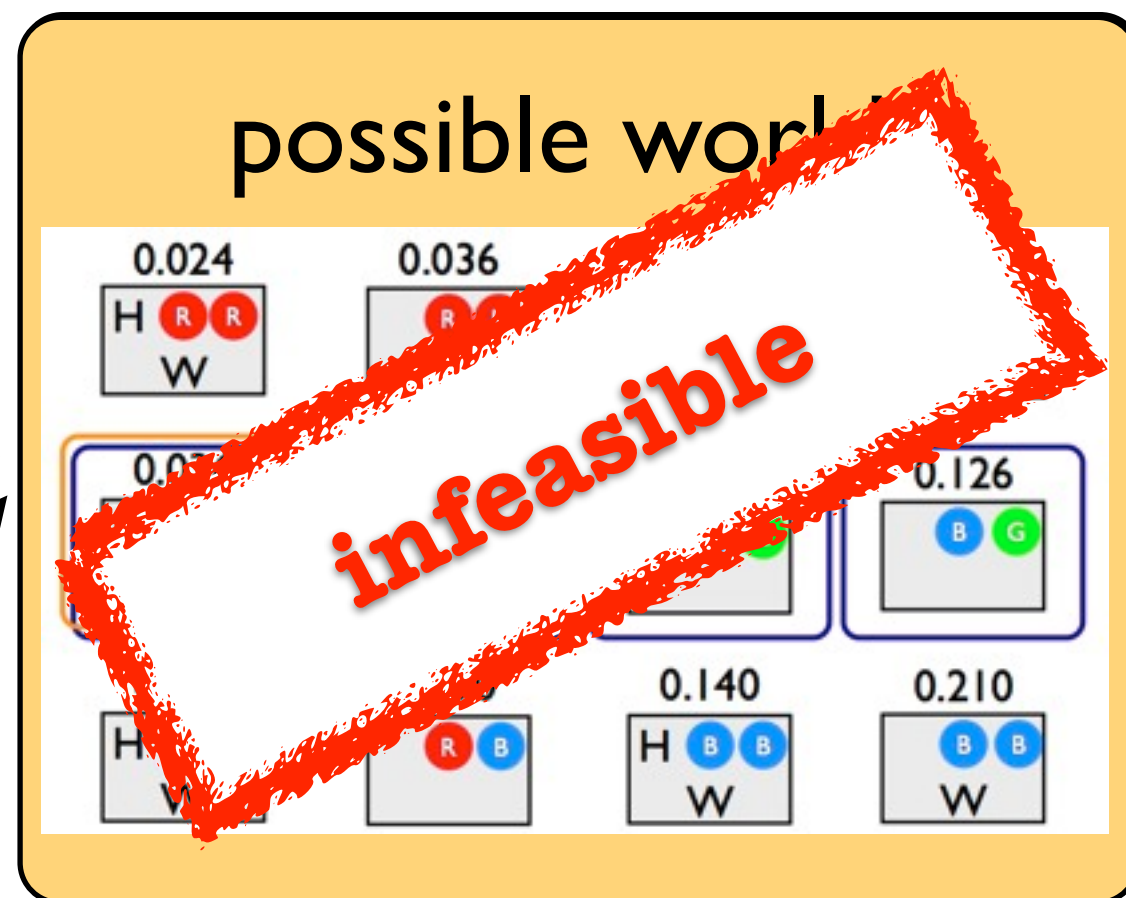
Answering Questions

Given:

program

queries

evidence



Find:

marginal
probabilities

conditional
probabilities

MPE state

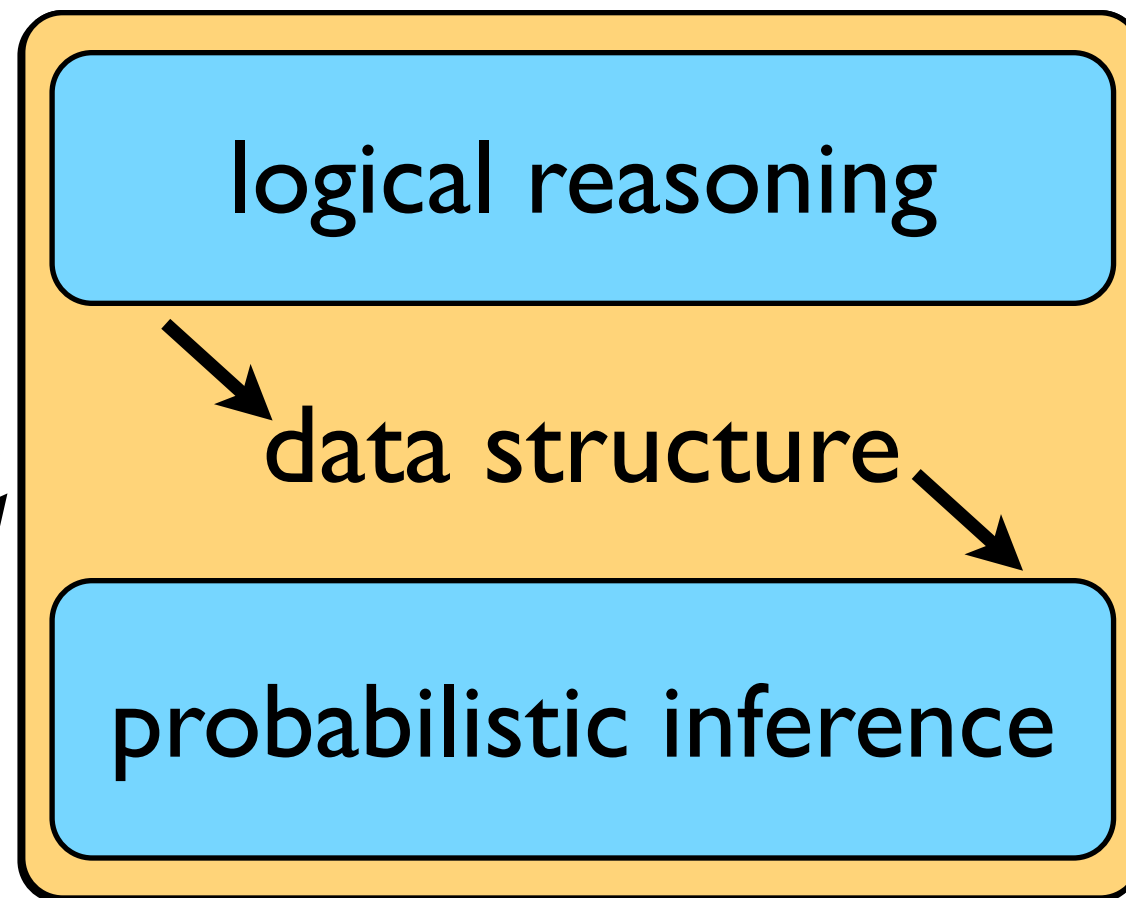
Answering Questions

Given:

program

queries

evidence



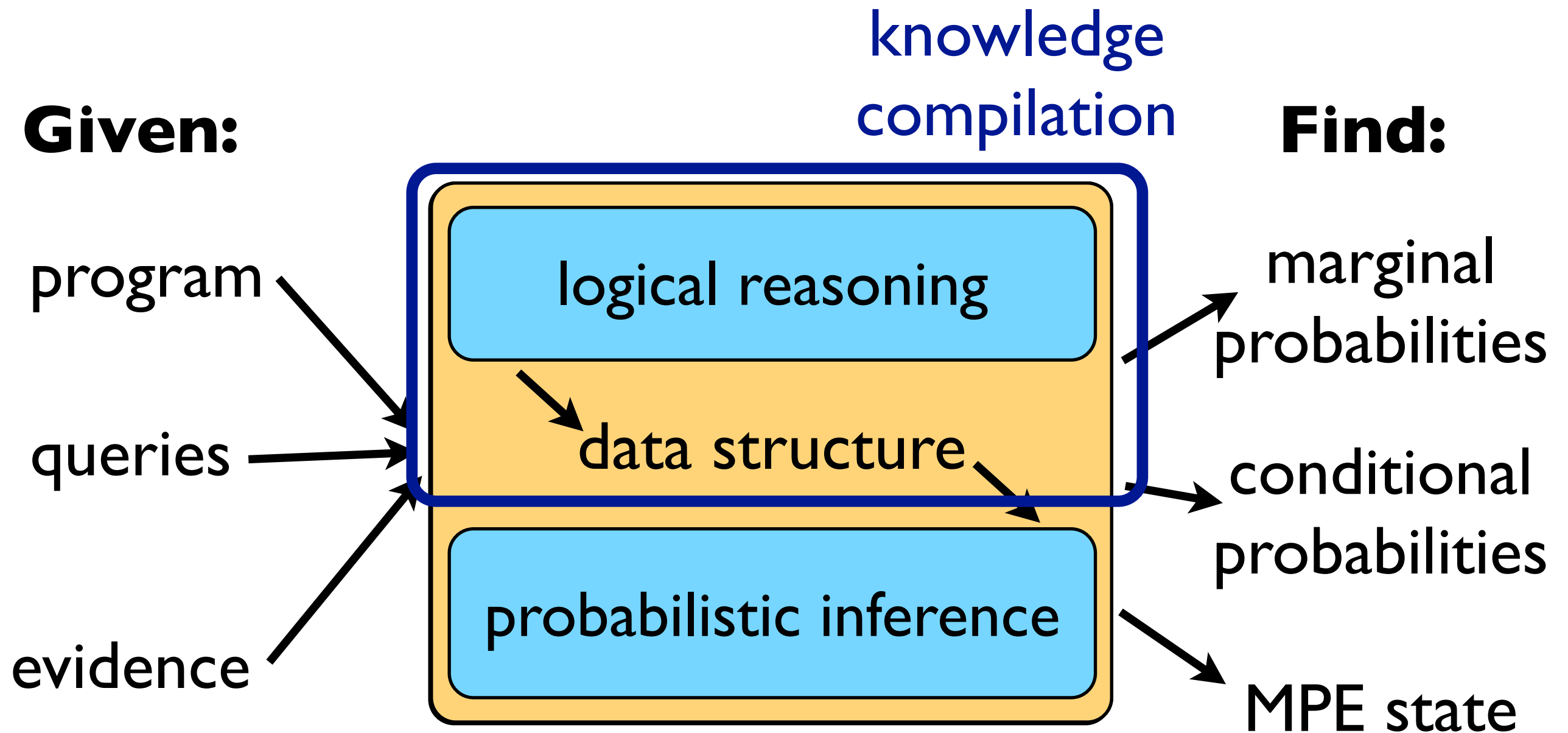
Find:

marginal
probabilities

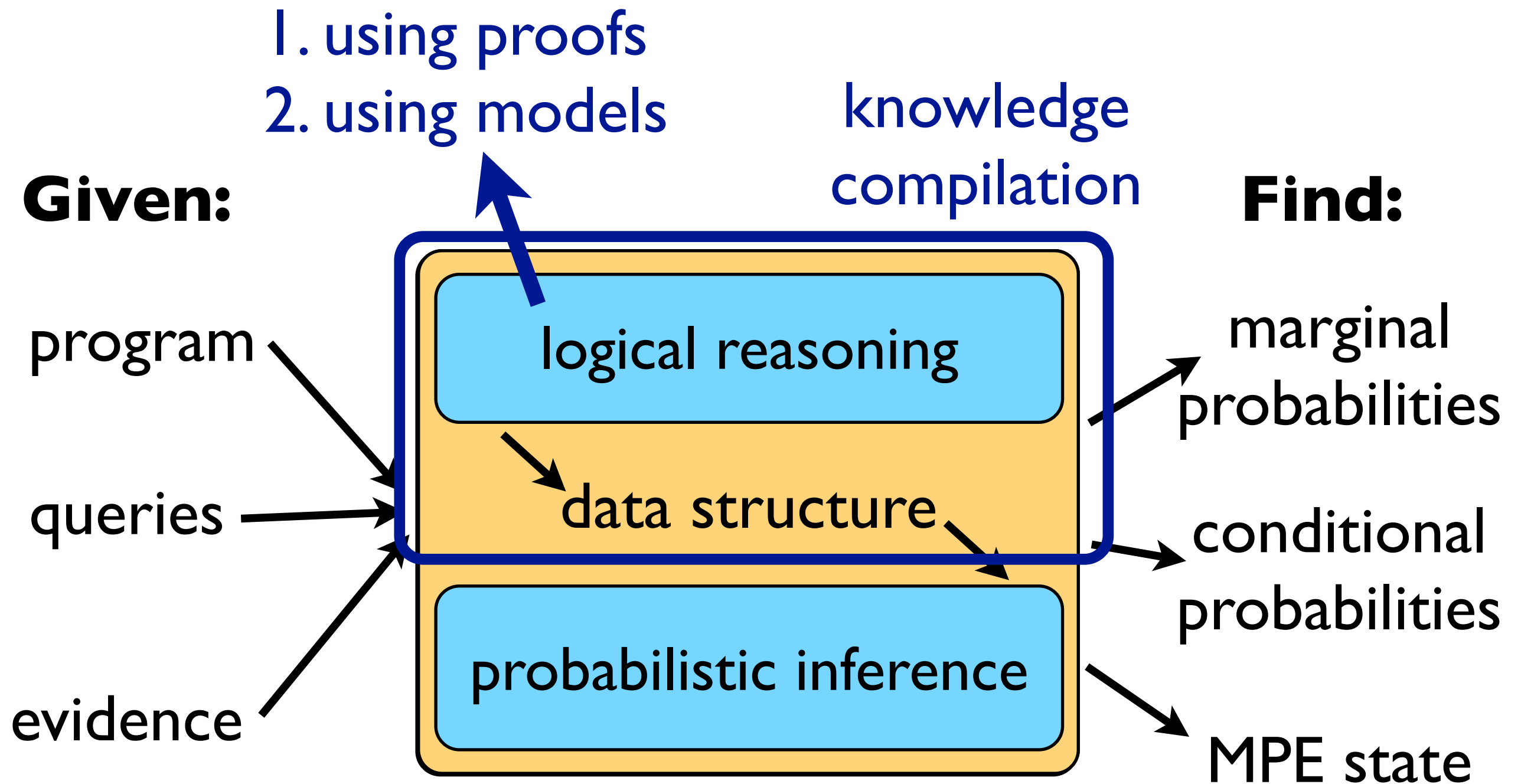
conditional
probabilities

MPE state

Answering Questions



Answering Questions



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .  
  
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

Logical Reasoning: Proofs in Prolog

```
?- smokes(carl) .
```

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

Logical Reasoning: Proofs in Prolog

`?- smokes(carl) .`

`?- stress(carl) .`



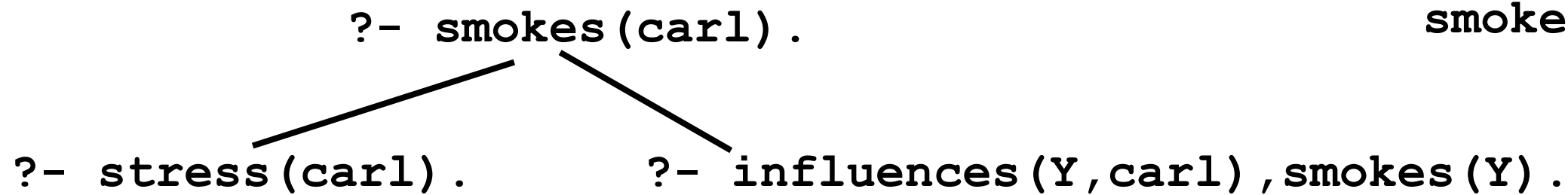
```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```


Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

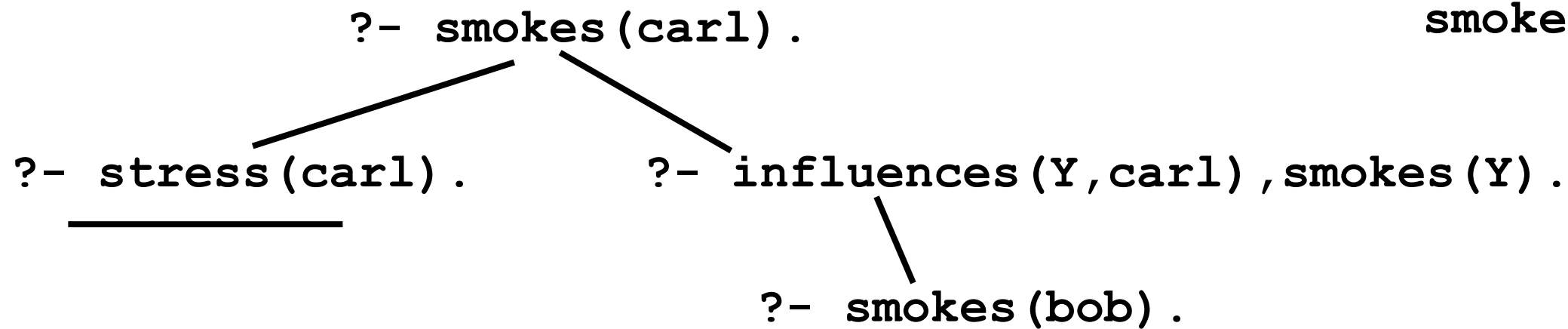
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

```
      ?- smokes(carl) .  
     /  \  
    /      \  
?- stress(carl) .    ?- influences(Y,carl) , smokes(Y) .  
  _____
```

Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

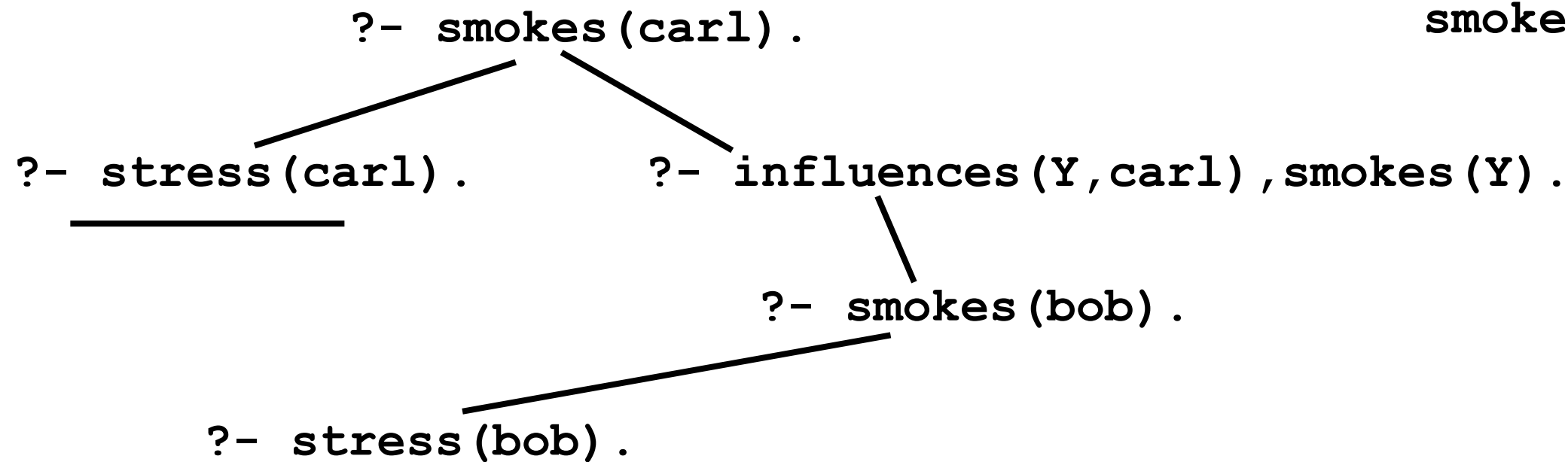
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

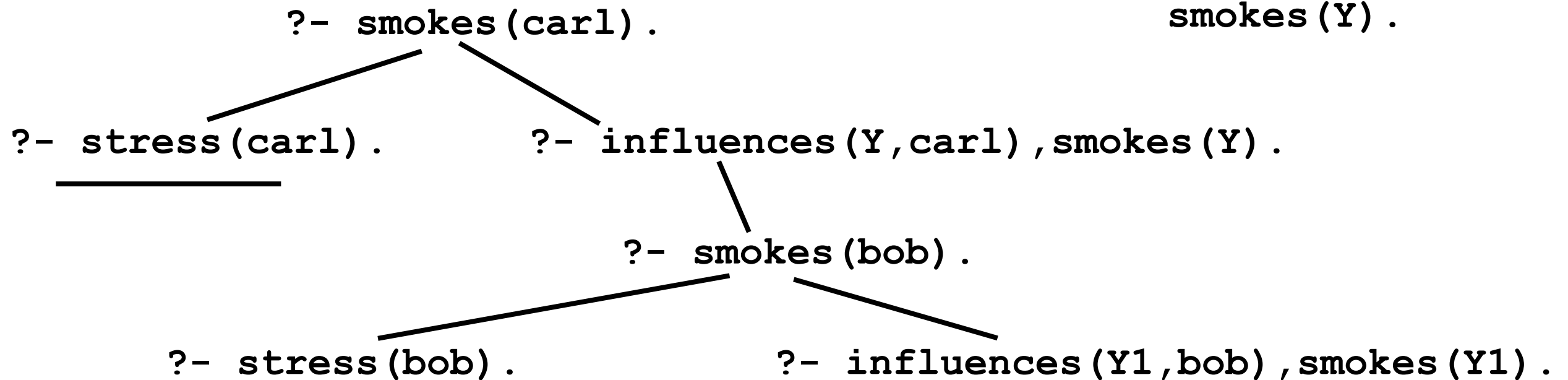
```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

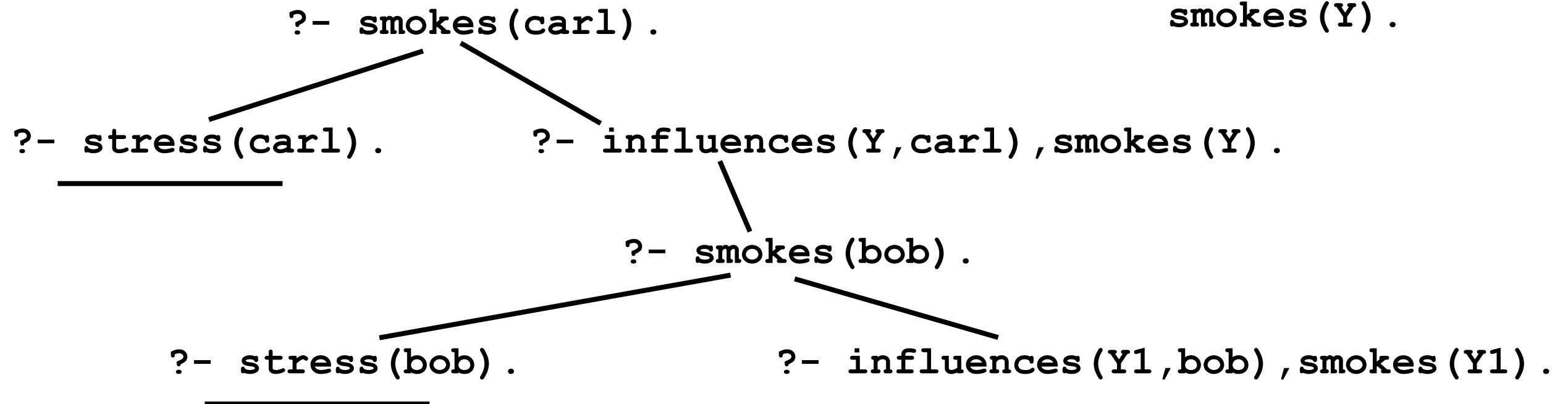
```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .  
  
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

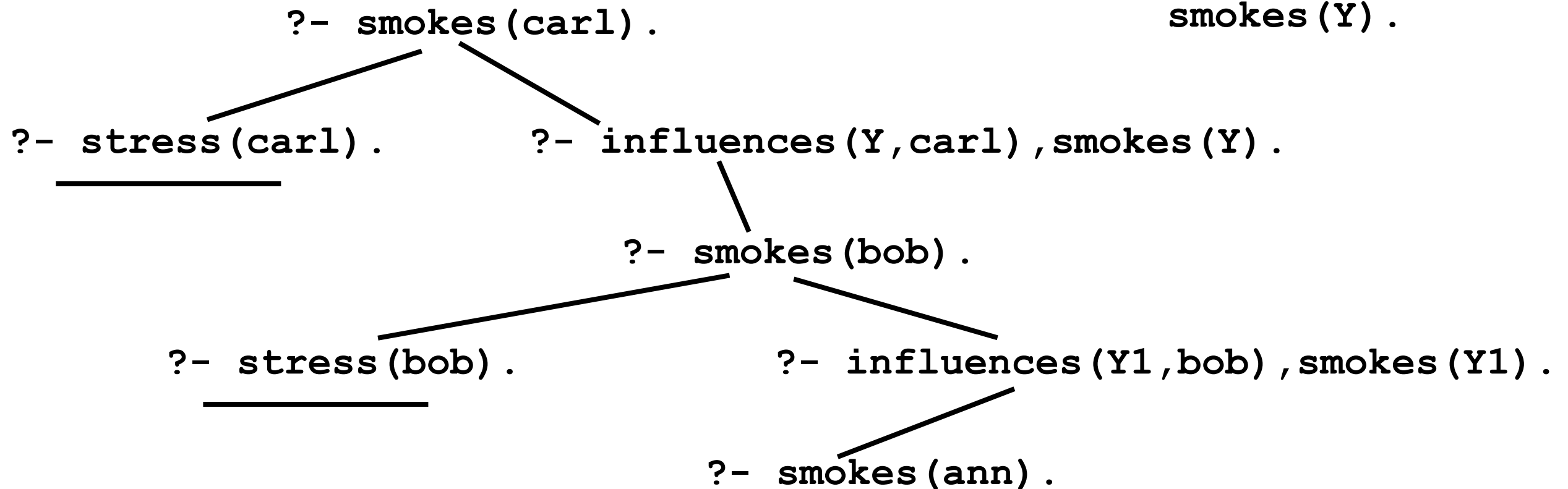
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

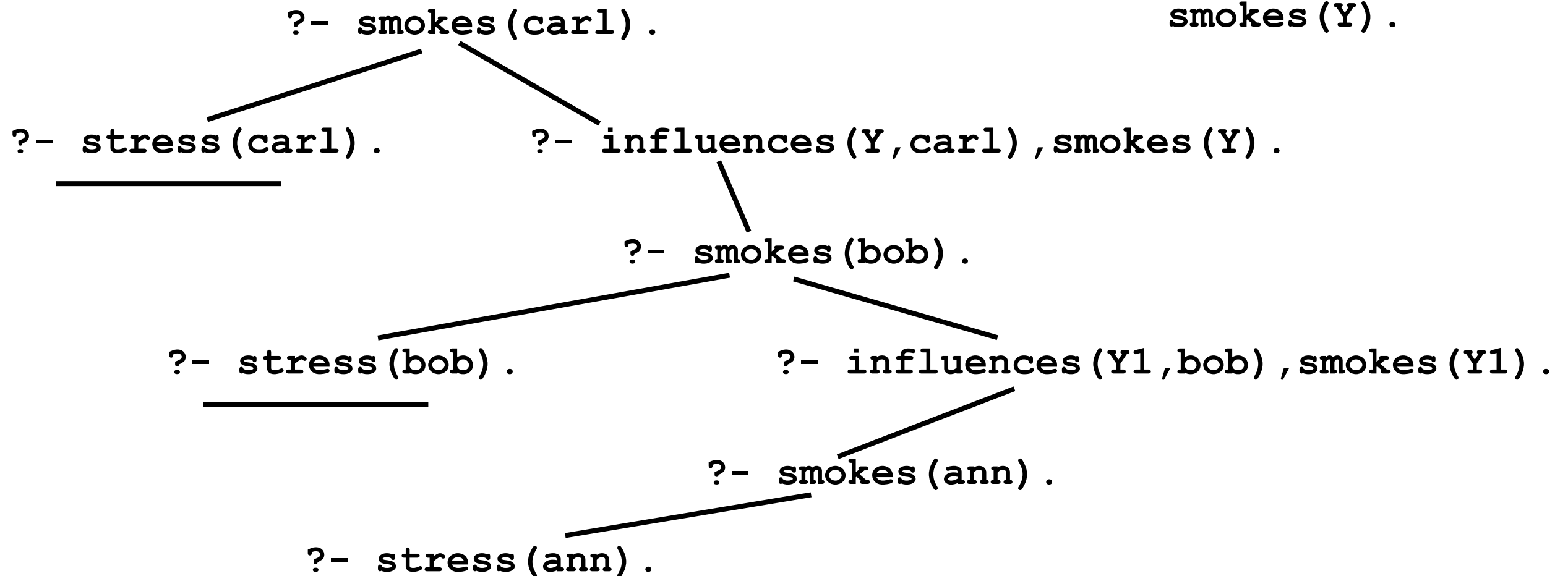
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

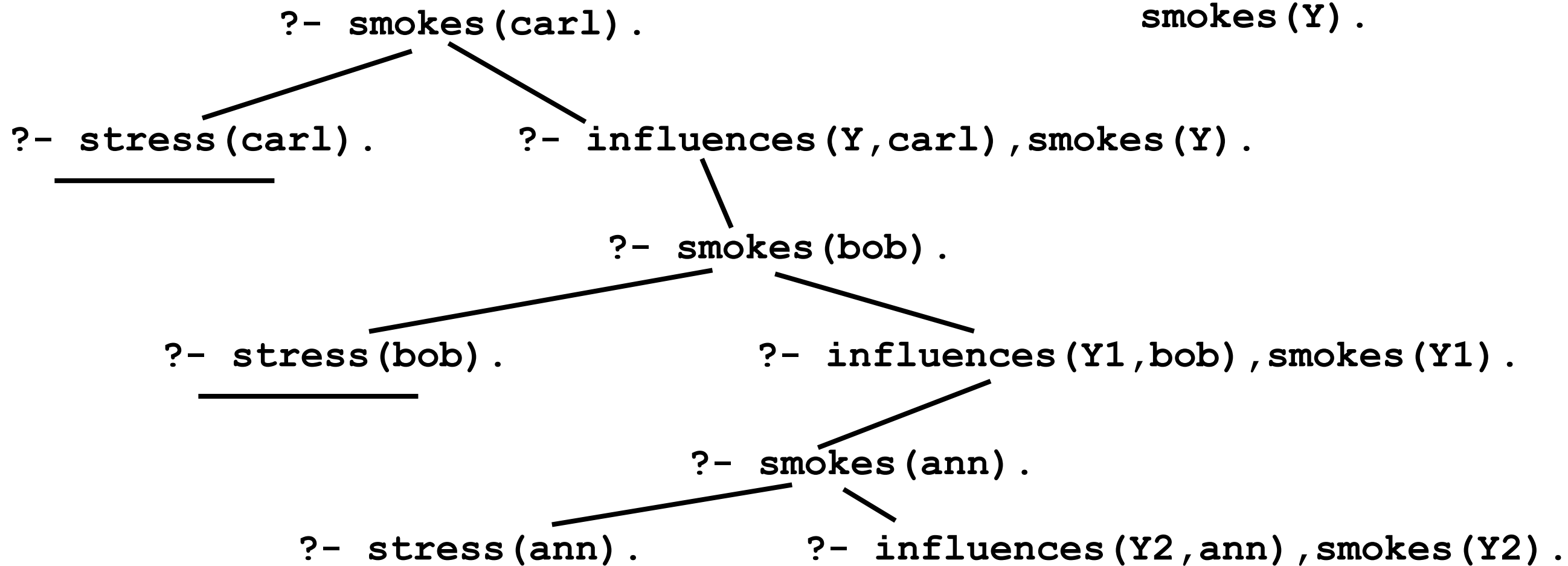
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

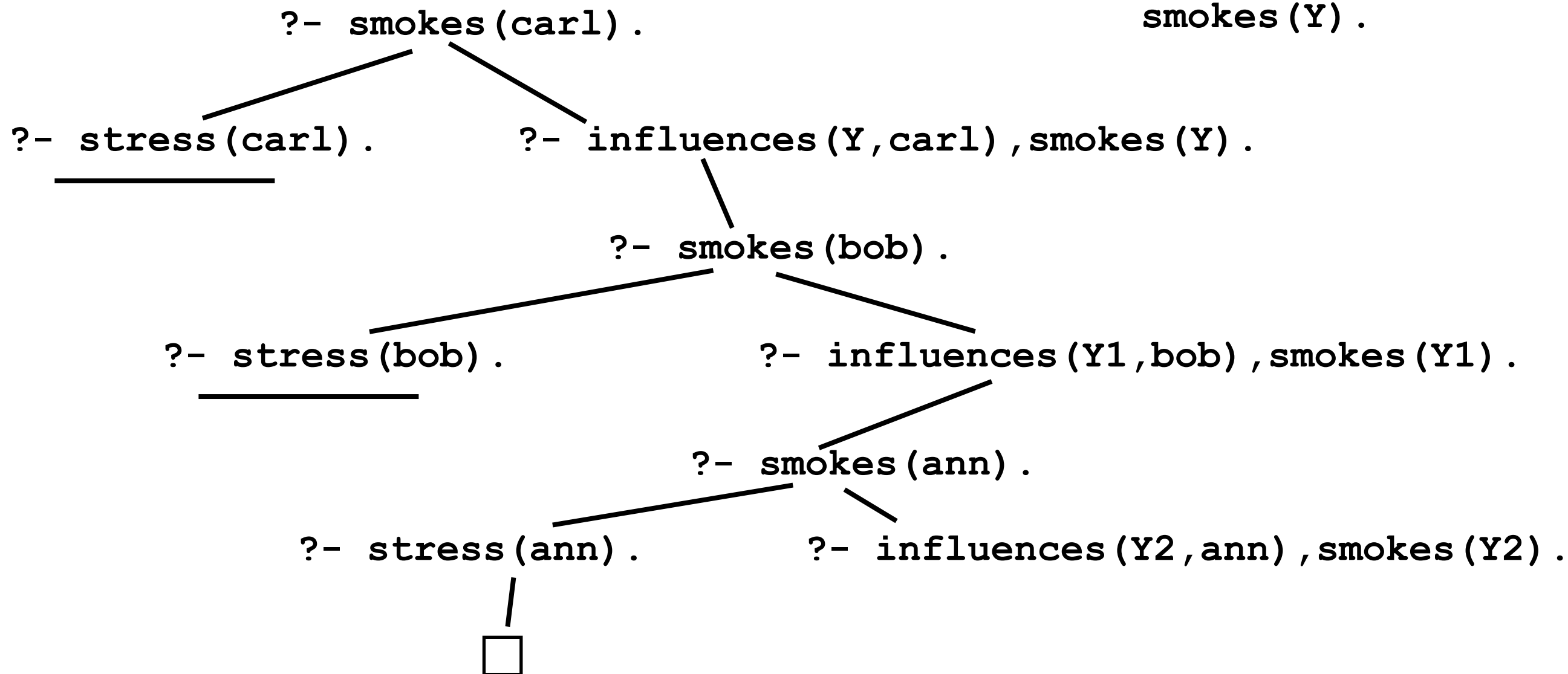
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

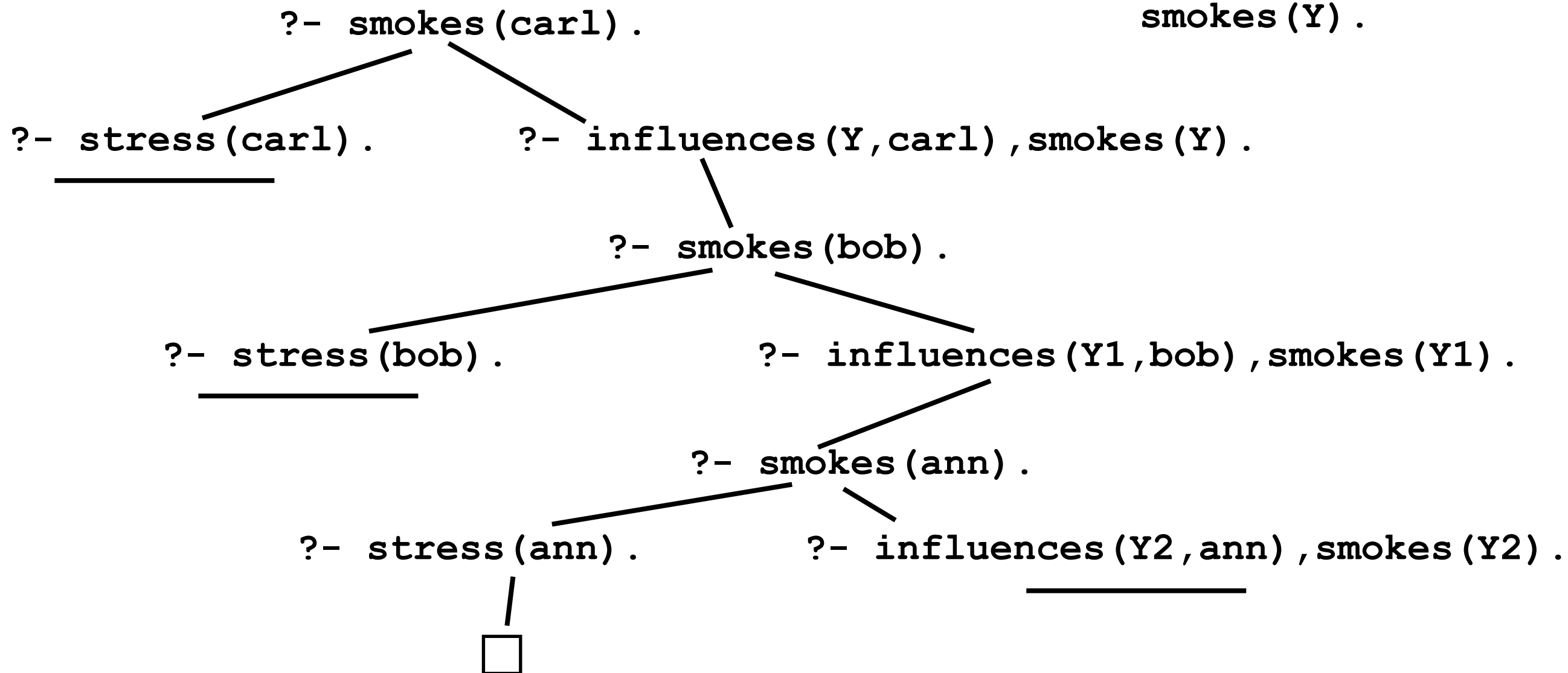
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

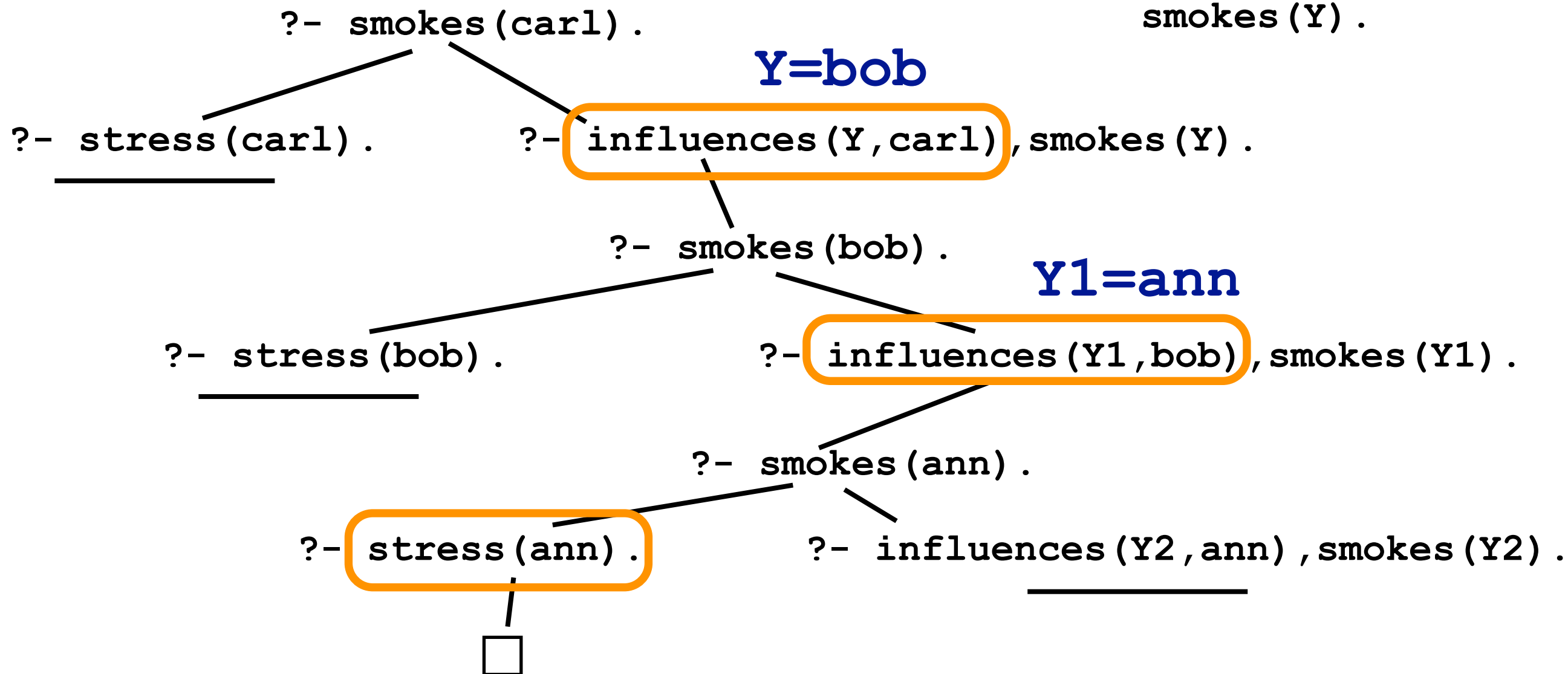
```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```



Logical Reasoning: Proofs in Prolog

```
stress(ann) .
influences(ann,bob) .
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .
smokes(X) :-
    influences(Y,X) ,
    smokes(Y) .
```

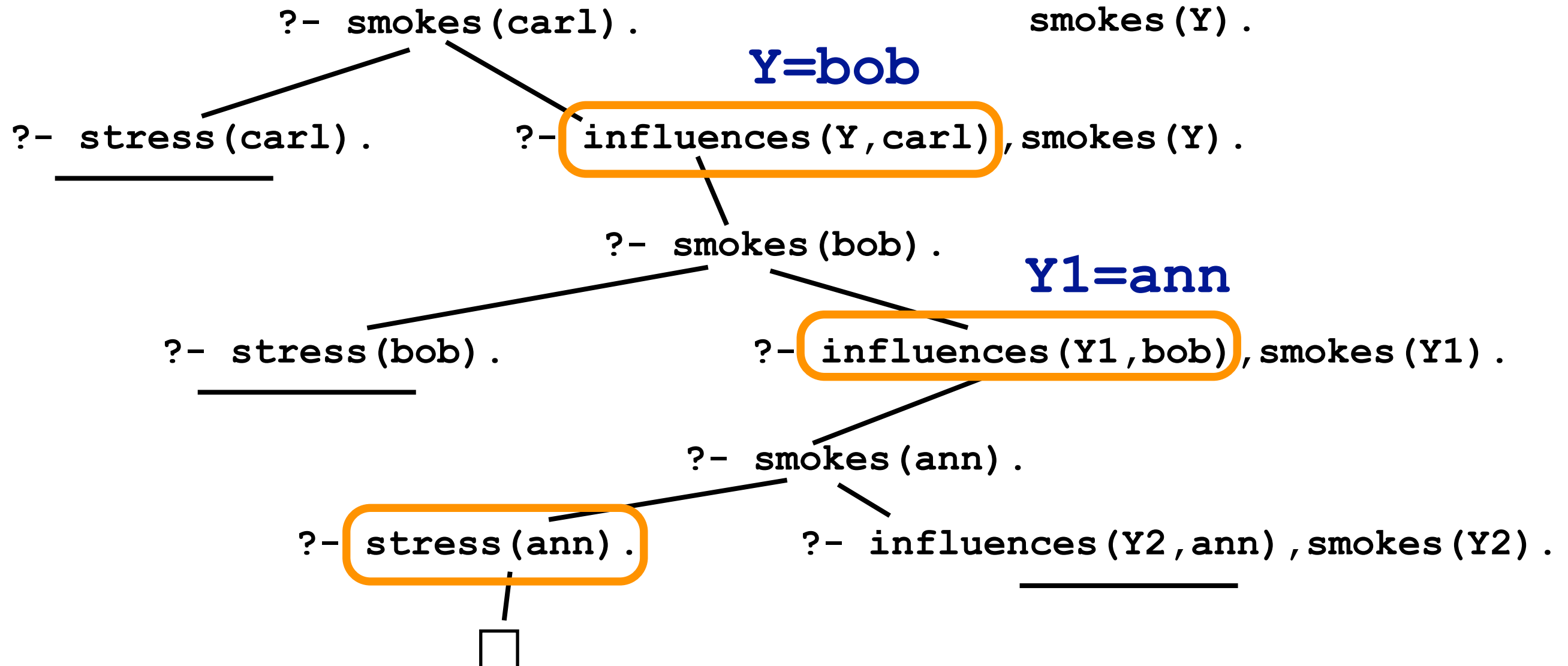


proof = facts used in successful derivation:
influences(bob,carl) & influences(ann,bob) & stress(ann)

Proofs in ProbLog

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

```
smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



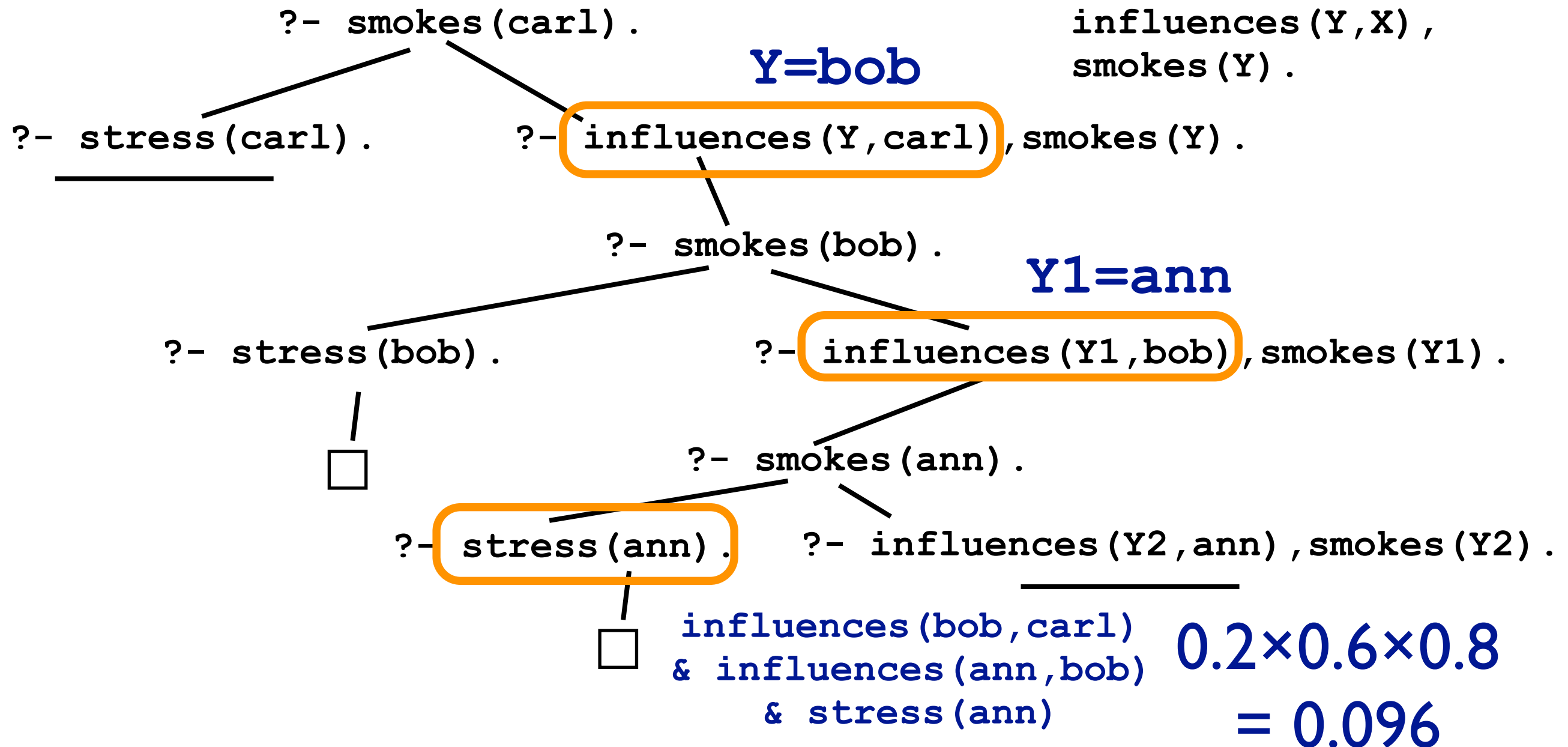
$\text{influences}(\text{bob}, \text{carl}) \ \& \ \text{influences}(\text{ann}, \text{bob}) \ \& \ \text{stress}(\text{ann})$

probability of proof = $0.2 \times 0.6 \times 0.8 = 0.096$

Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

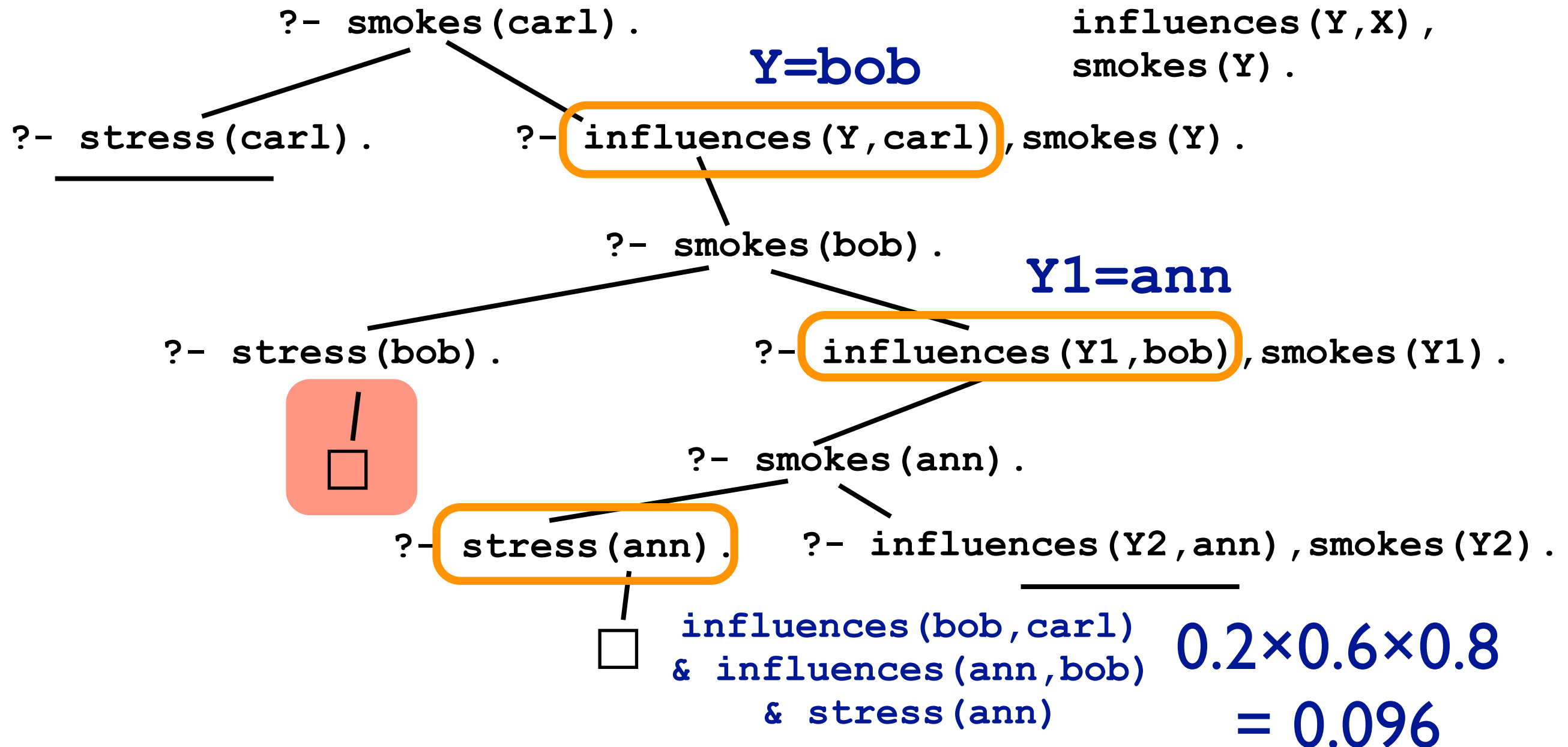
```
smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

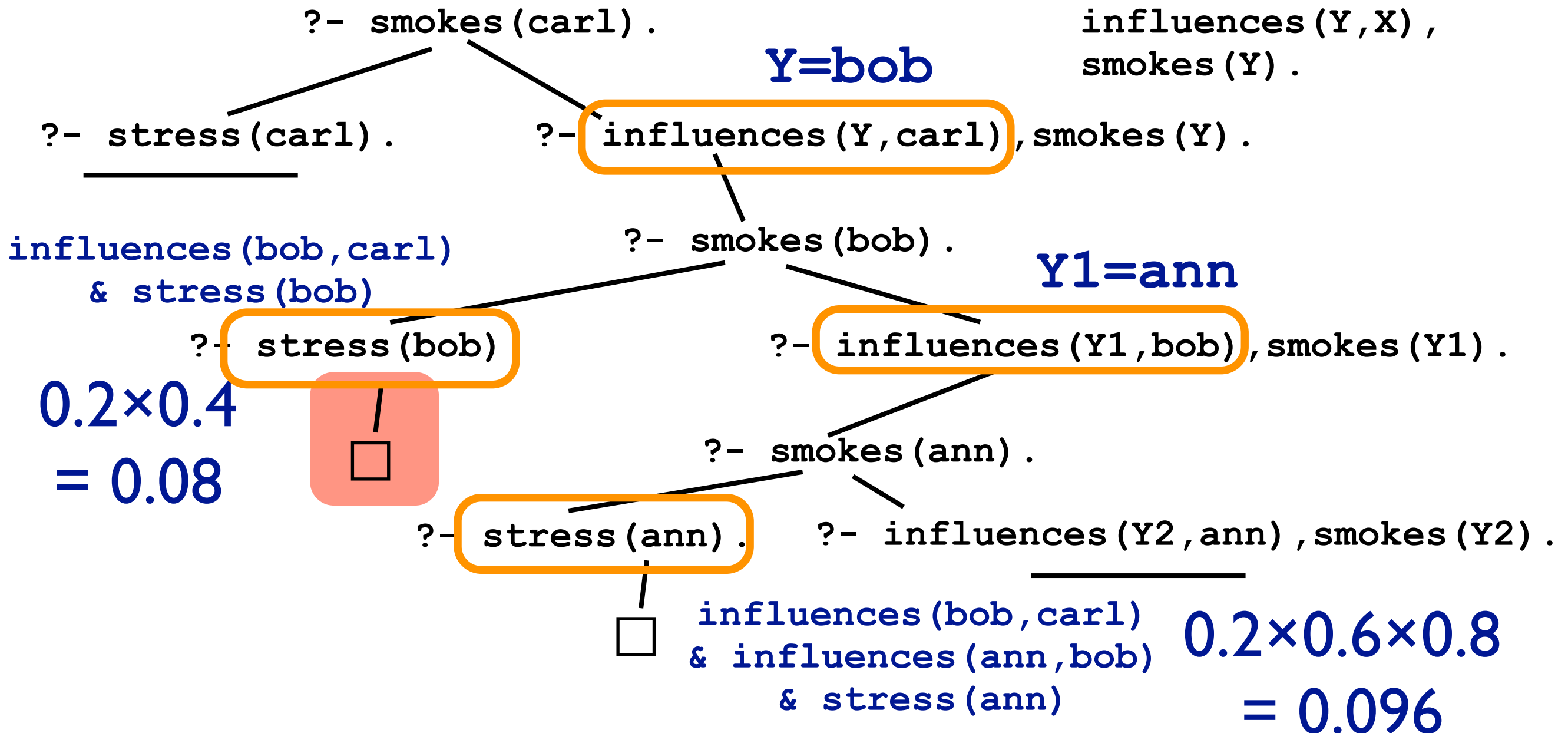
```
smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

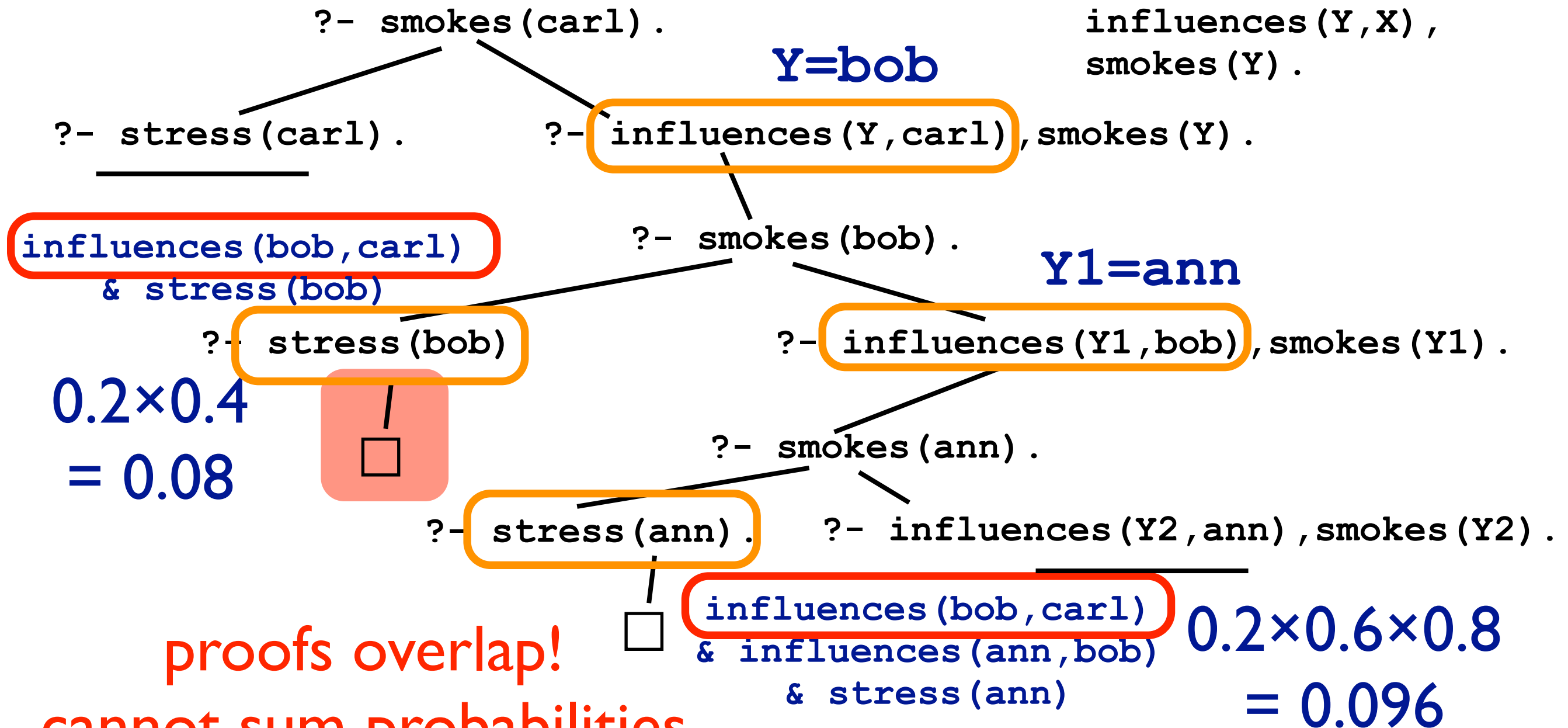
```
smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

```
smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



proofs overlap!
cannot sum probabilities
(disjoint-sum-problem)

Disjoint-Sum-Problem

possible worlds

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`

`...`

Disjoint-Sum-Problem

possible worlds

`influences(bob,carl) &
influences(ann,bob) & stress(ann)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`

...

Disjoint-Sum-Problem

possible worlds

`influences(bob,carl) &
influences(ann,bob) & stress(ann)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`

`... influences(bob,carl) & stress(bob)`

Disjoint-Sum-Problem

possible worlds

`influences(bob,carl) &
influences(ann,bob) & stress(ann)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`

`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`

`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`

`... influences(bob,carl) & stress(bob)`

sum of proof probabilities: $0.096 + 0.08 = 0.1760$

Disjoint-Sum-Problem

possible worlds

`influences(bob,carl) &
influences(ann,bob) & stress(ann)`

<code>infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)</code>	0.0576
<code>infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)</code>	0.0384
<code>infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)</code>	0.0256
<code>infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)</code>	0.0096
<code>infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)</code>	0.0064

... `influences(bob,carl) & stress(bob)` $\Sigma = 0.1376$

sum of proof probabilities: $0.096 + 0.08 = 0.1760$

Disjoint-Sum-Problem

possible worlds

solution: knowledge compilation

<code>infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)</code>	0.0576
<code>infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)</code>	0.0384
<code>infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)</code>	0.0256
<code>infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)</code>	0.0096
<code>infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)</code>	0.0064

... `influences(bob,carl) & stress(bob)` $\Sigma = 0.1376$

sum of proof probabilities: $0.096 + 0.08 = 0.1760$

Binary Decision Diagrams [Bryant 86]

- compact graphical representation of Boolean formula
- popular in many branches of CS
- automatically disjoins proofs
→ efficient probability computation

Binary Decision Diagrams

[Bryant 86]

$$X \vee Y \vee Z$$

Binary Decision Diagrams

[Bryant 86]

$$X \vee Y \vee Z$$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

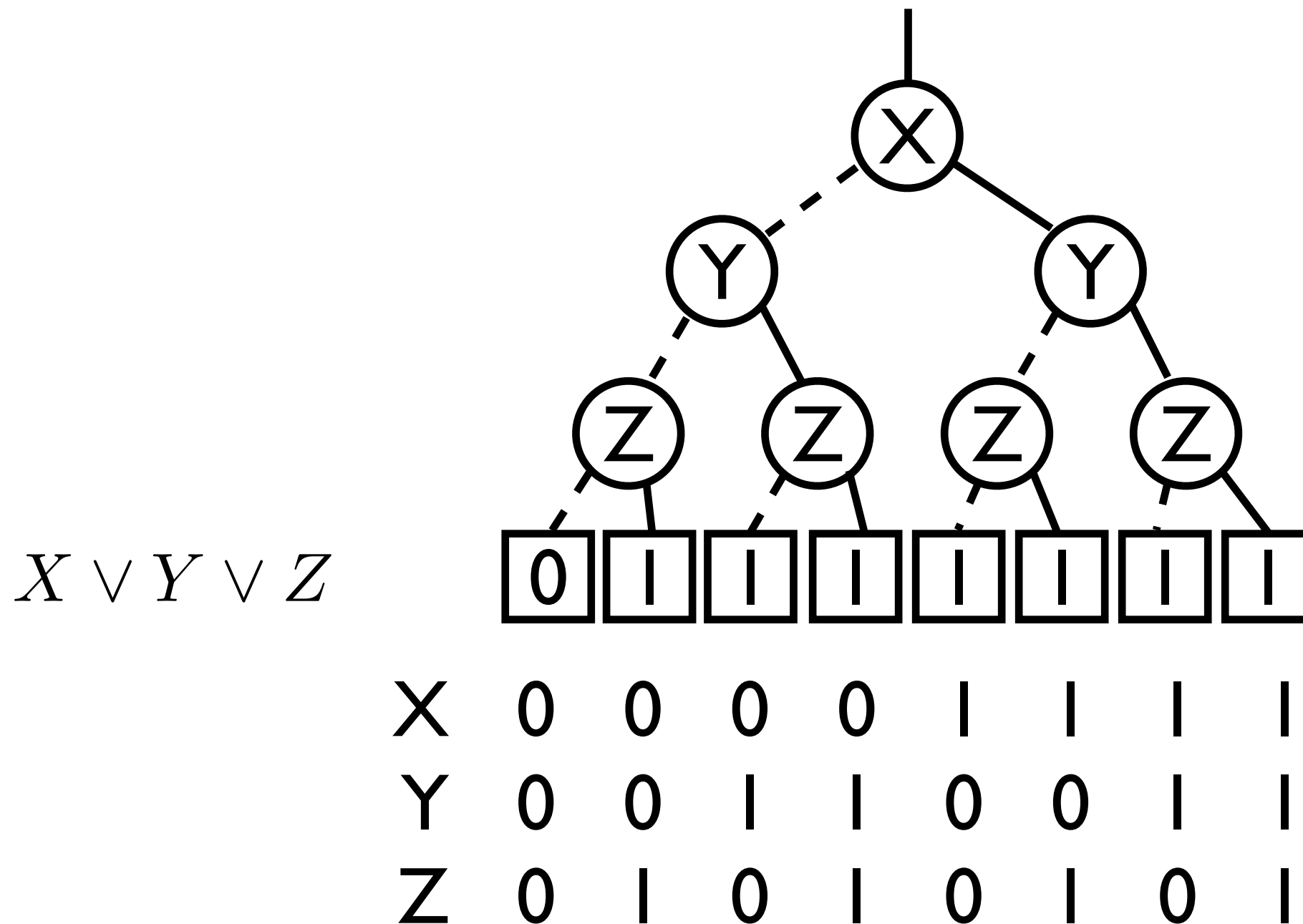
[Bryant 86]

$X \vee Y \vee Z$

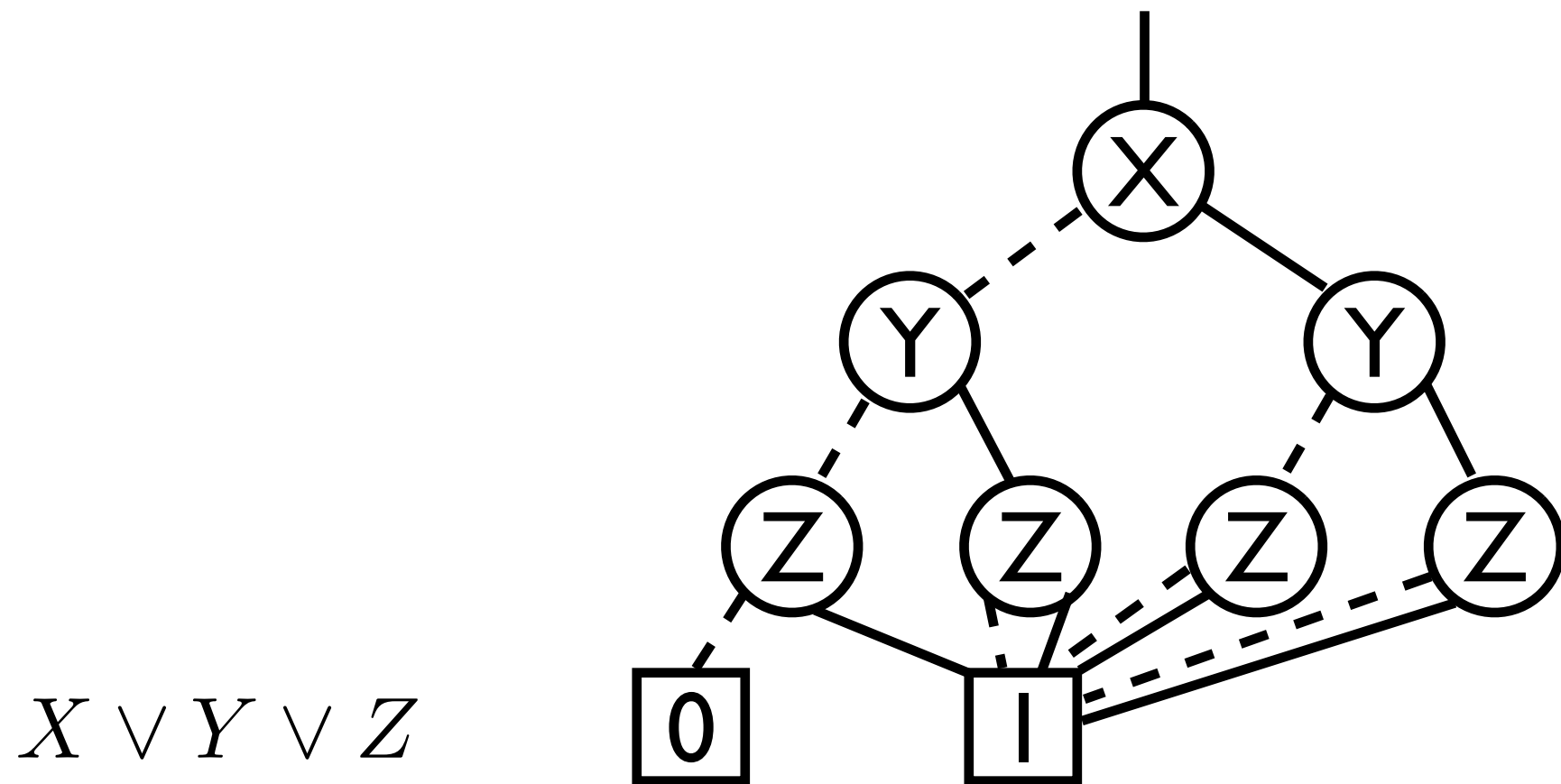
X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1



Binary Decision Diagrams [Bryant 86]

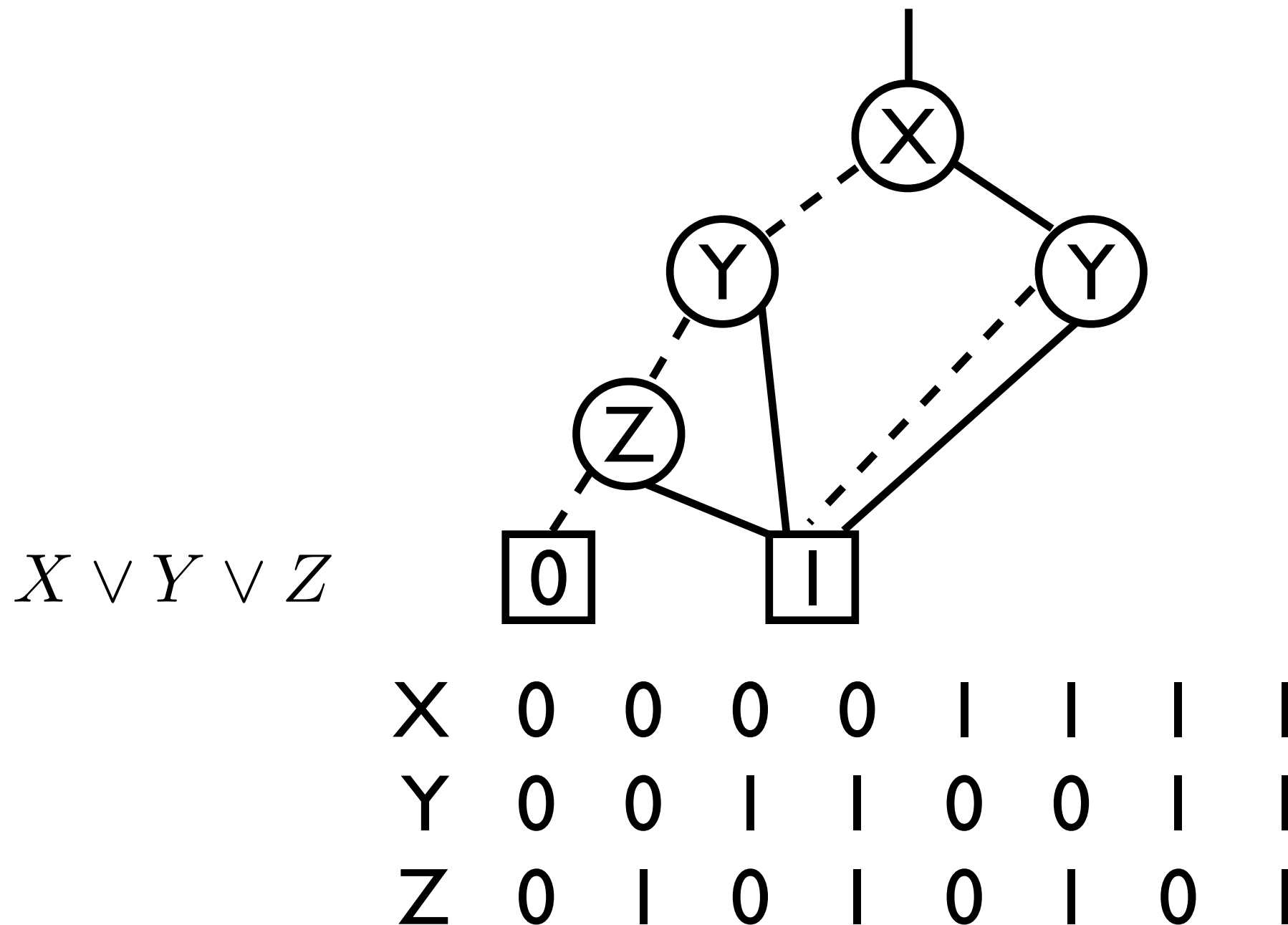


Binary Decision Diagrams [Bryant 86]

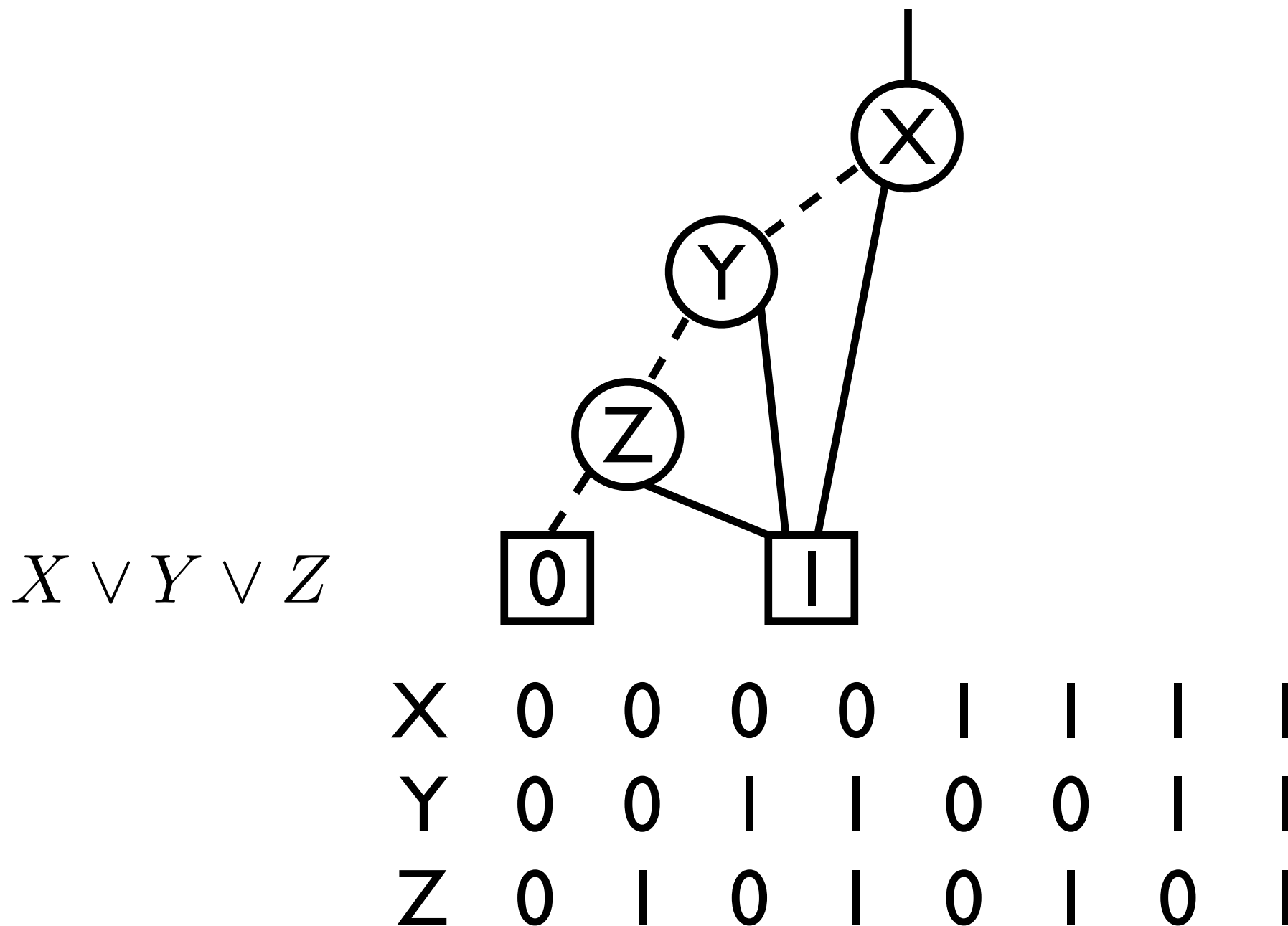


X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

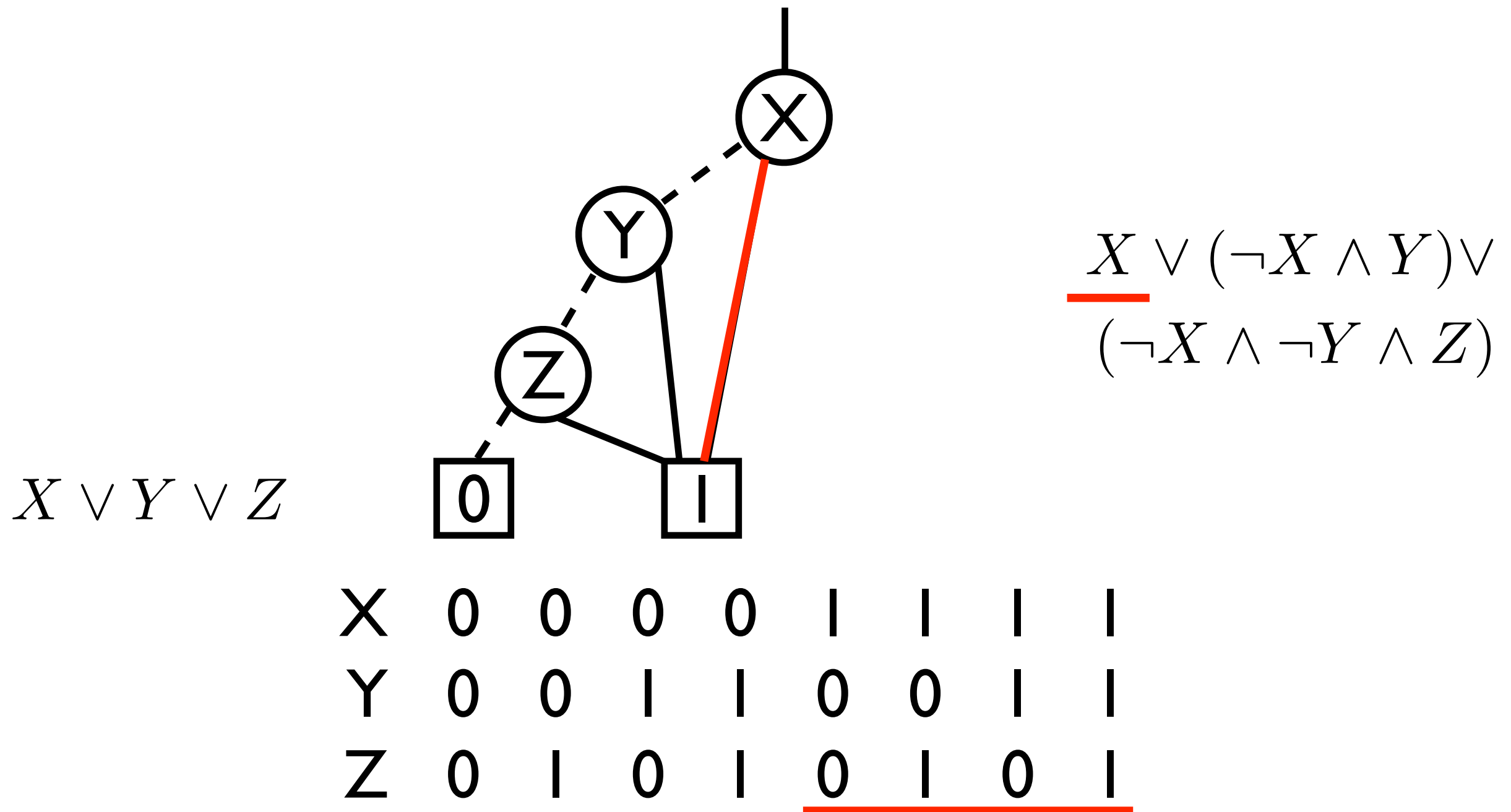
Binary Decision Diagrams [Bryant 86]



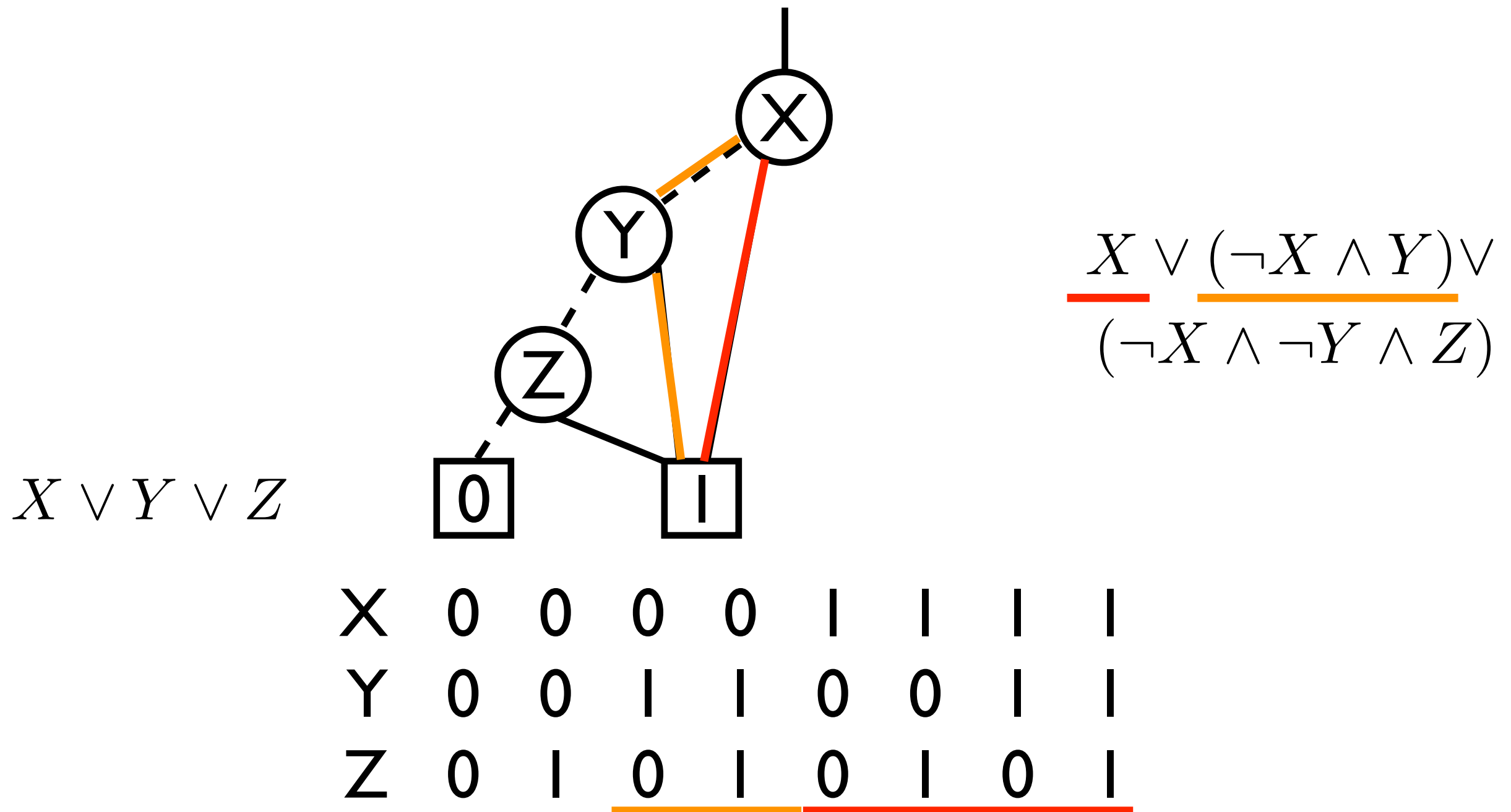
Binary Decision Diagrams [Bryant 86]



Binary Decision Diagrams [Bryant 86]

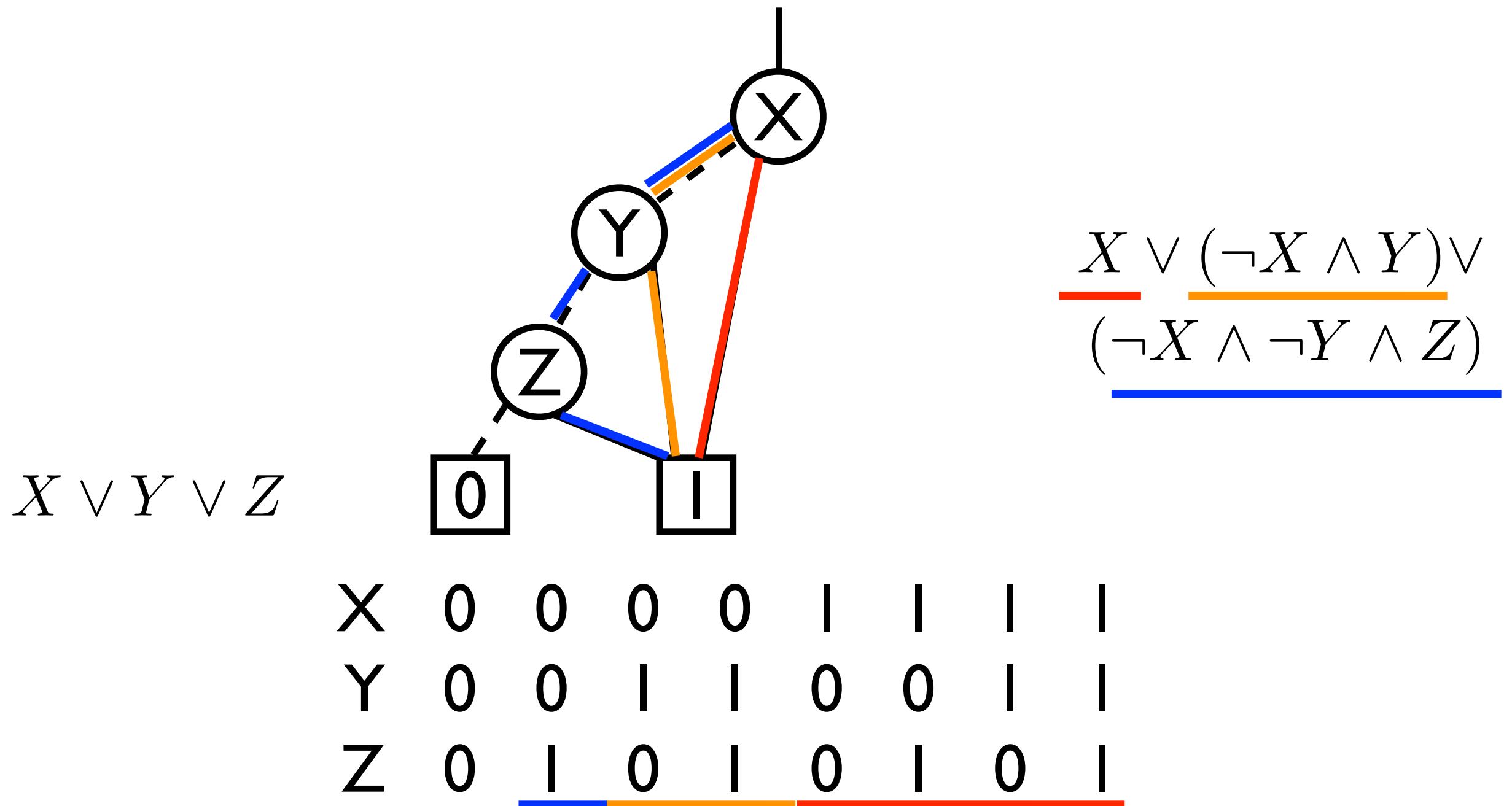


Binary Decision Diagrams [Bryant 86]

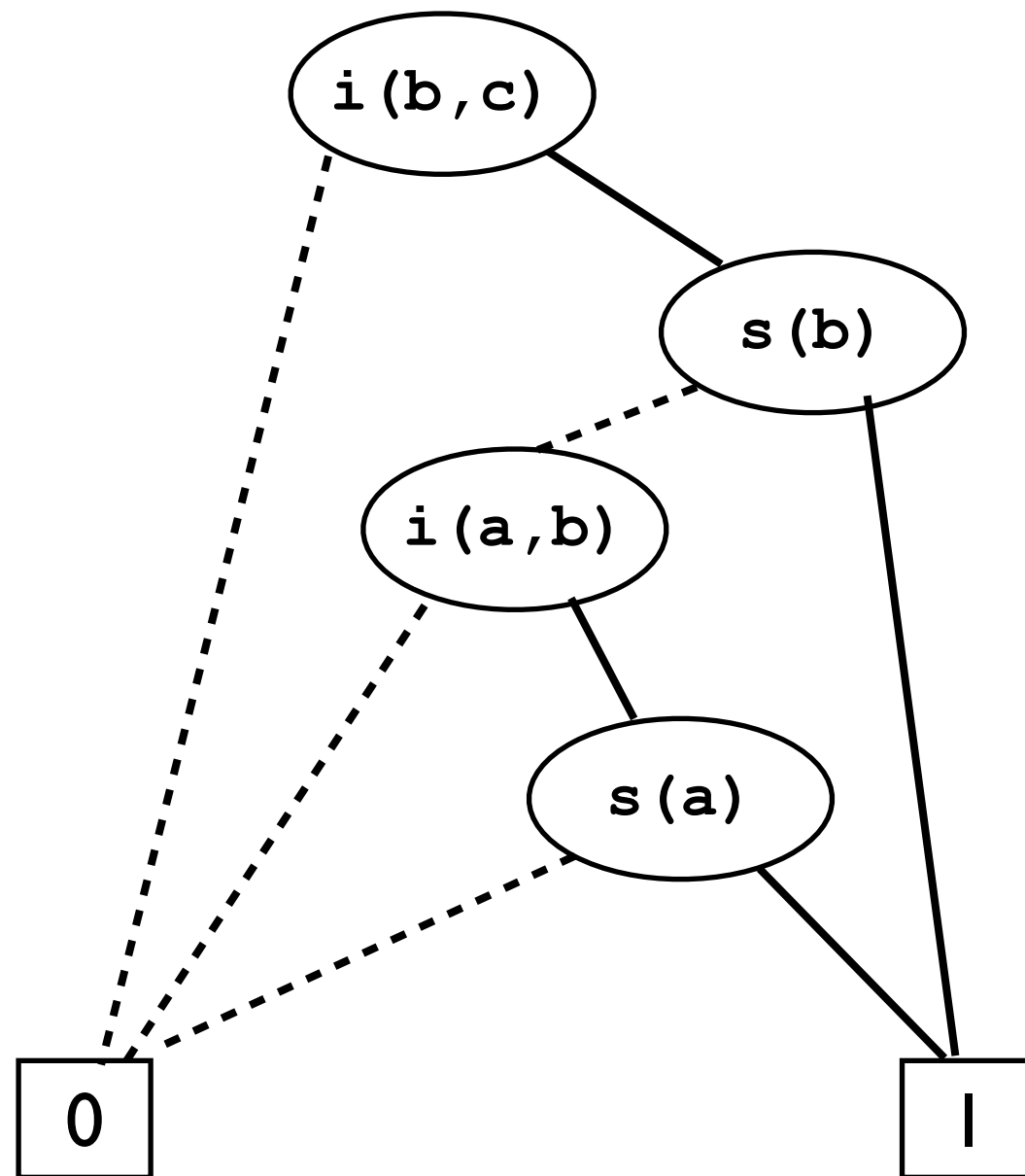


Binary Decision Diagrams

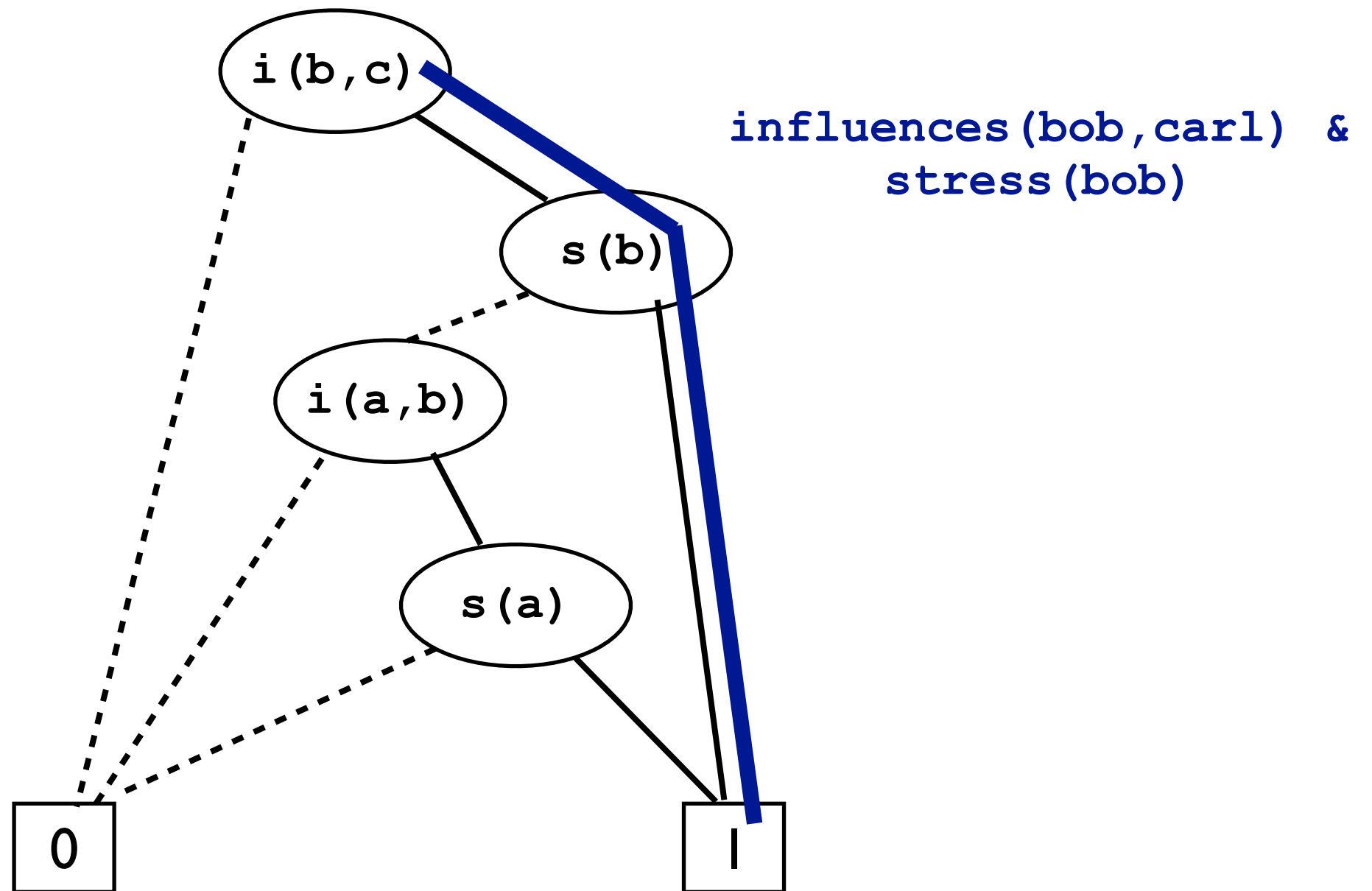
[Bryant 86]



Binary Decision Diagrams [Bryant 86]

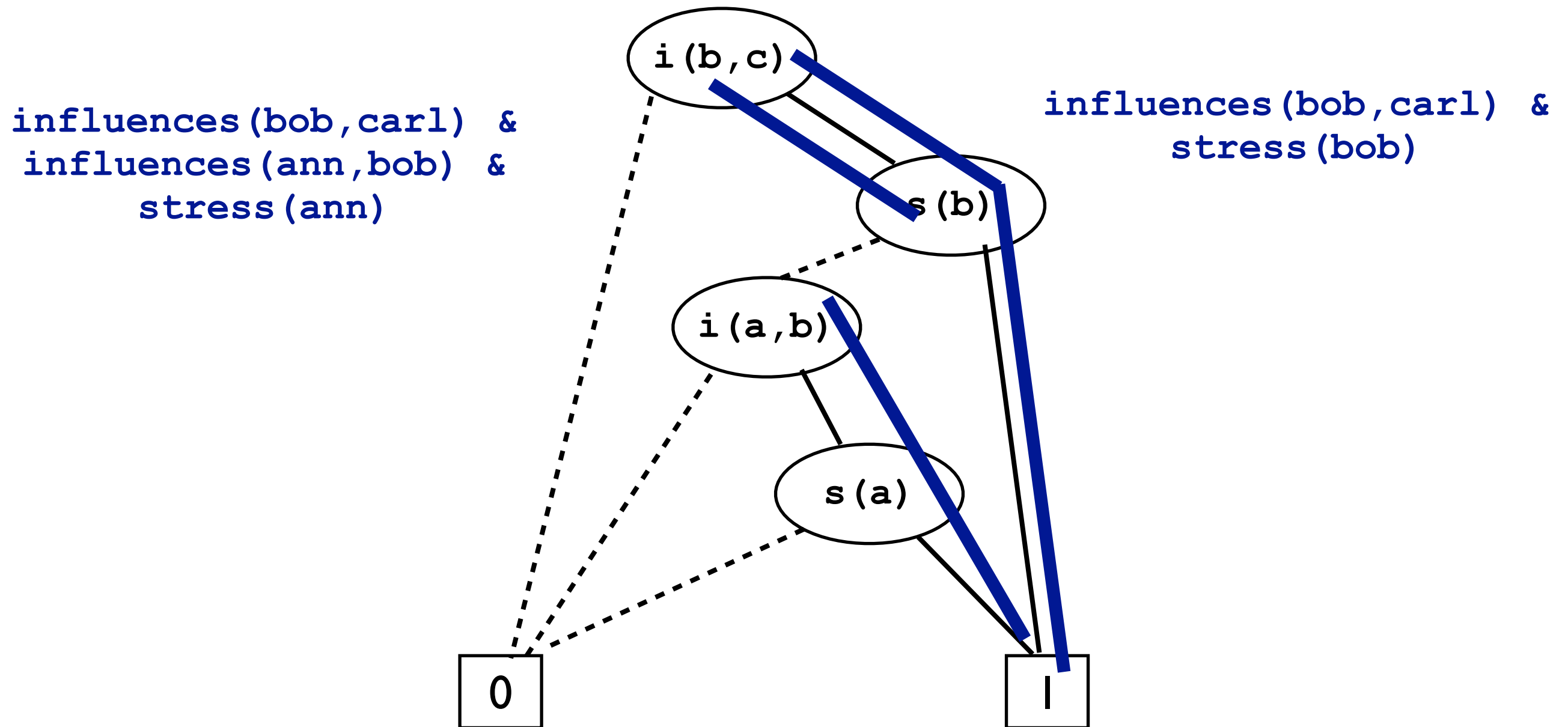


Binary Decision Diagrams [Bryant 86]

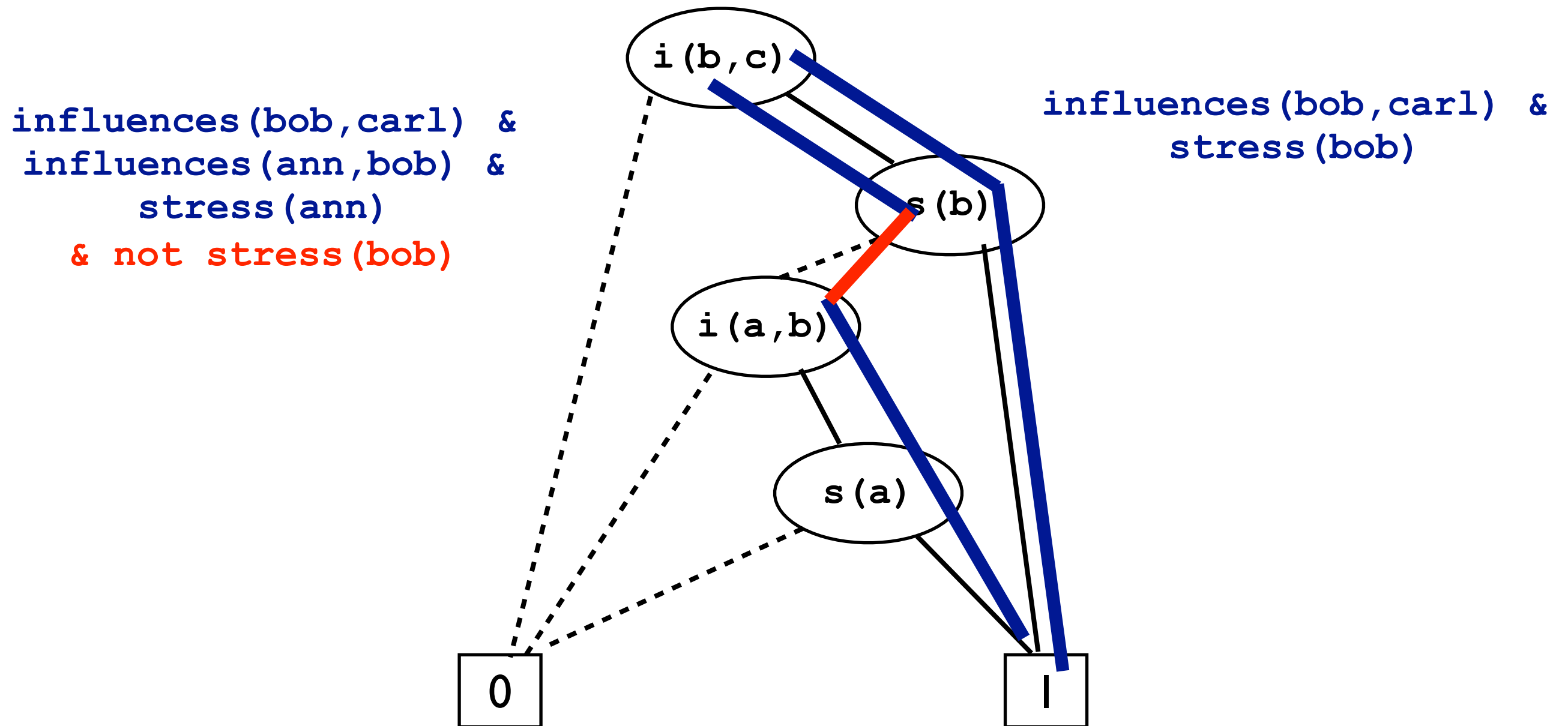


Binary Decision Diagrams

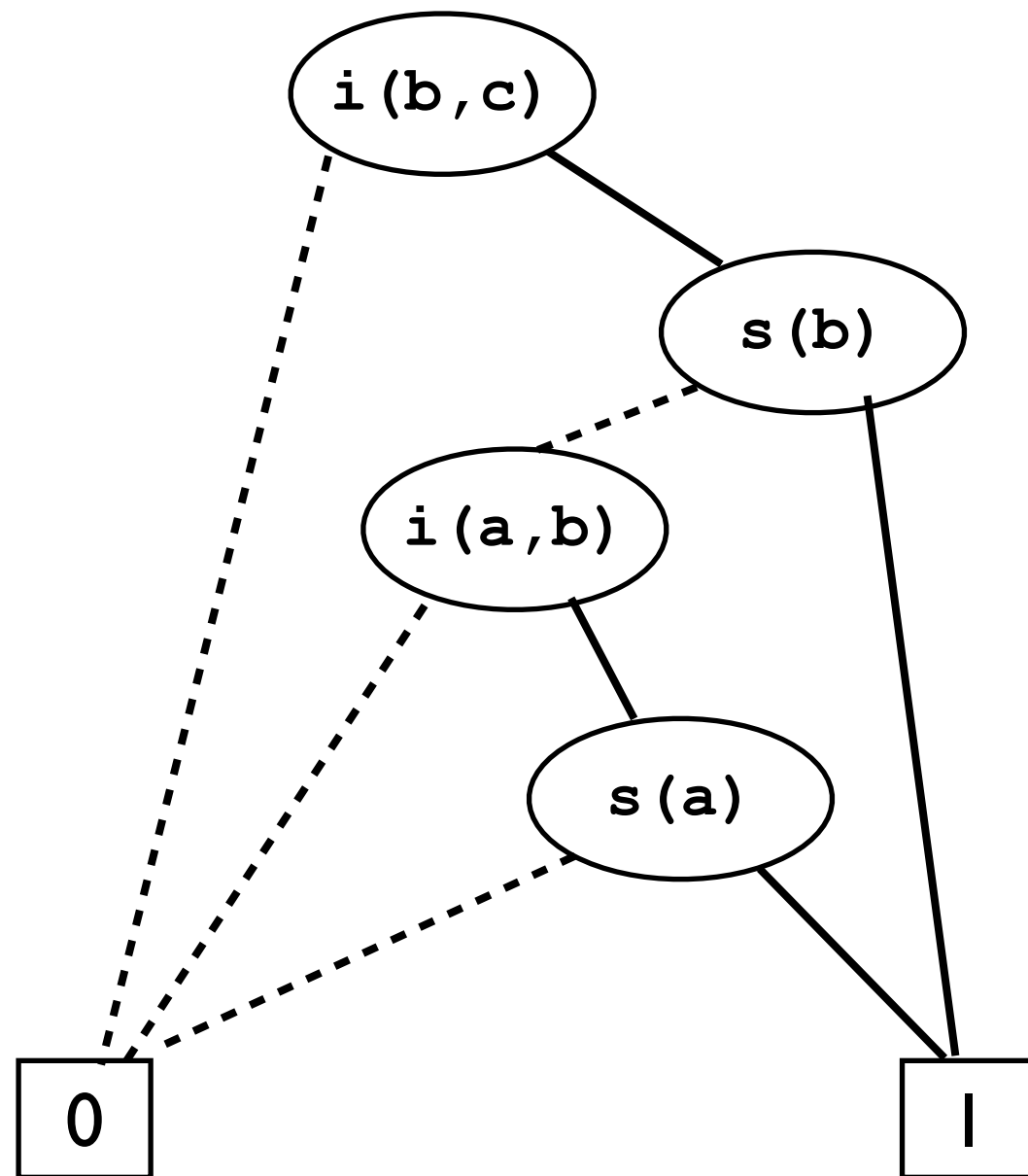
[Bryant 86]



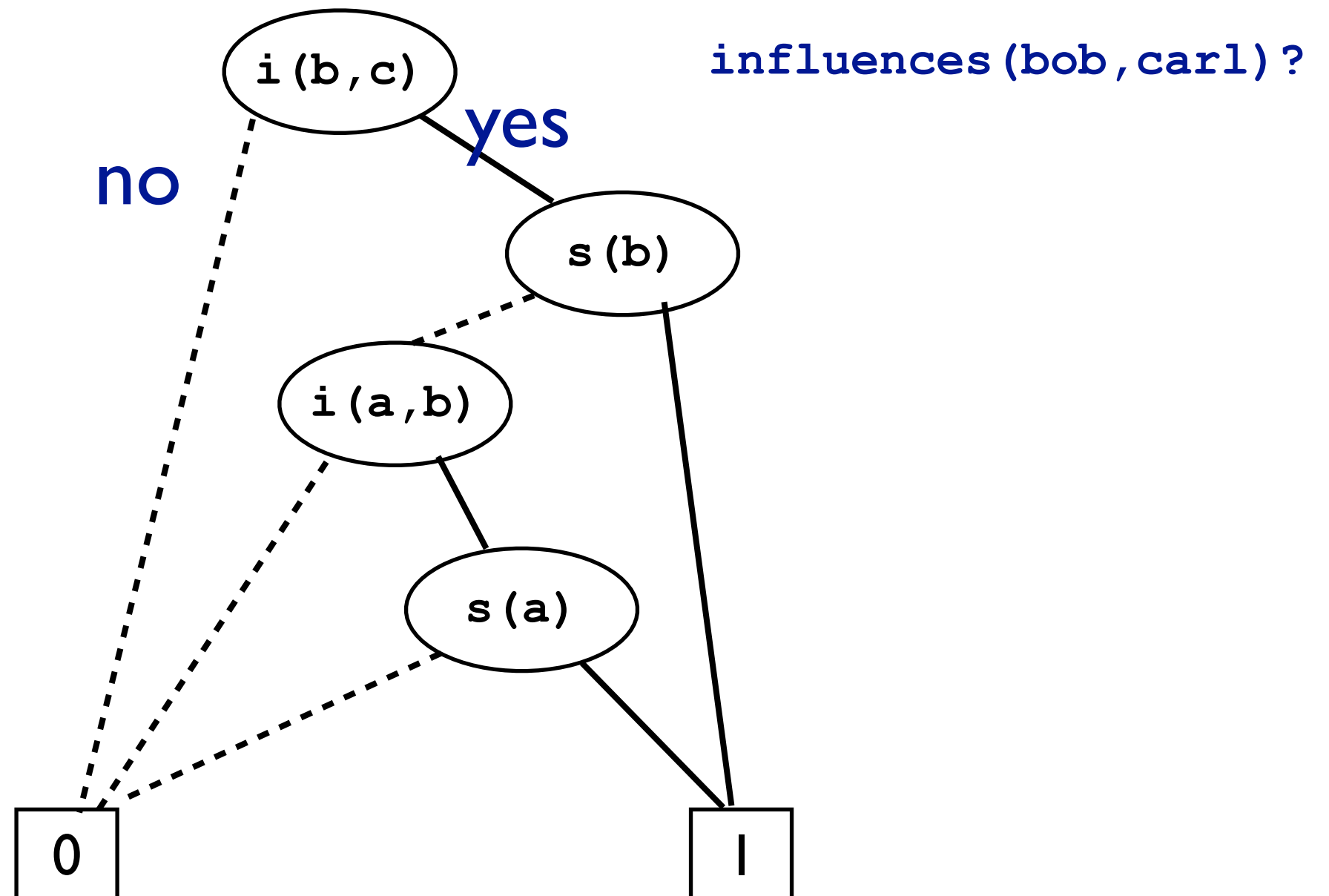
Binary Decision Diagrams [Bryant 86]



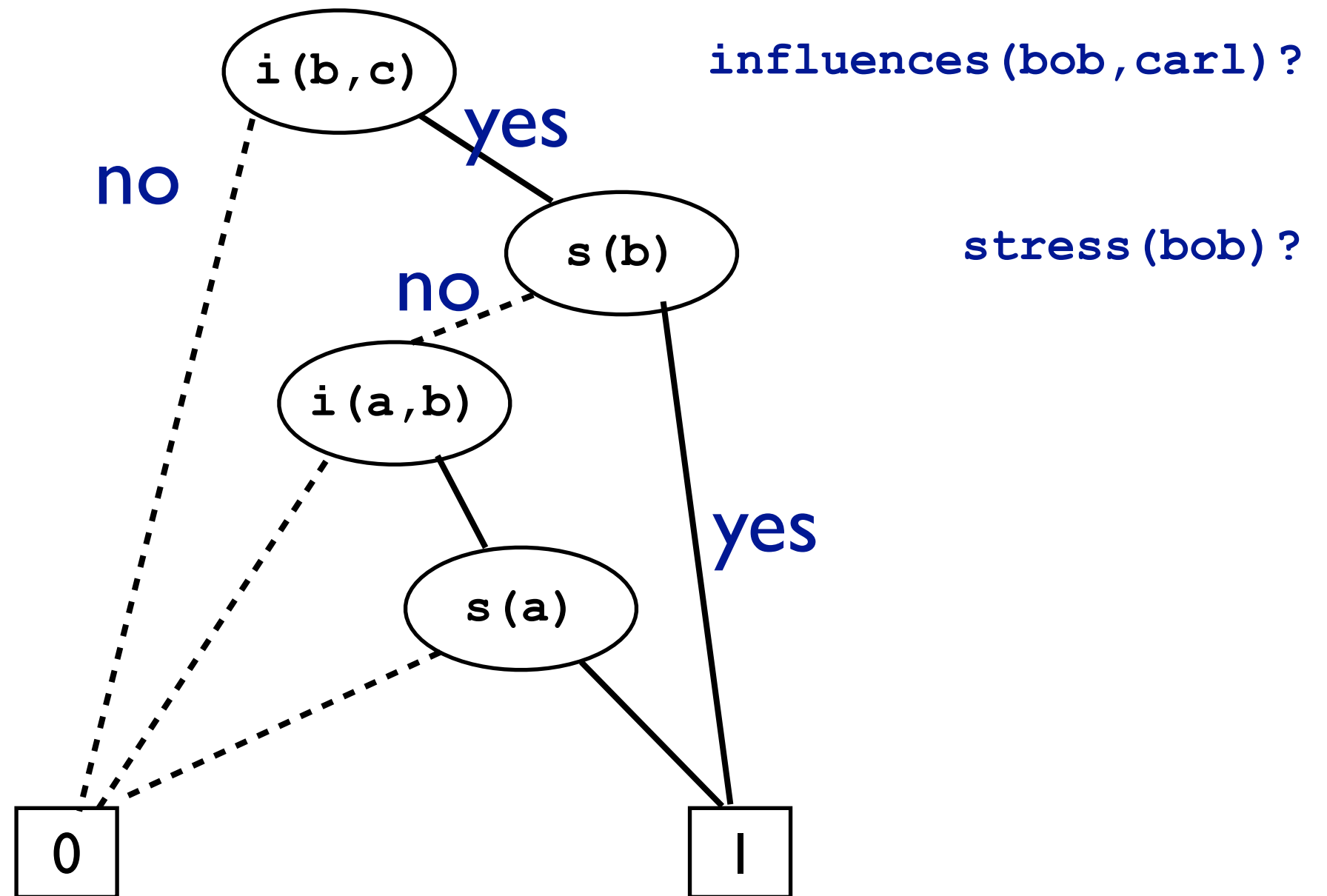
Binary Decision Diagrams



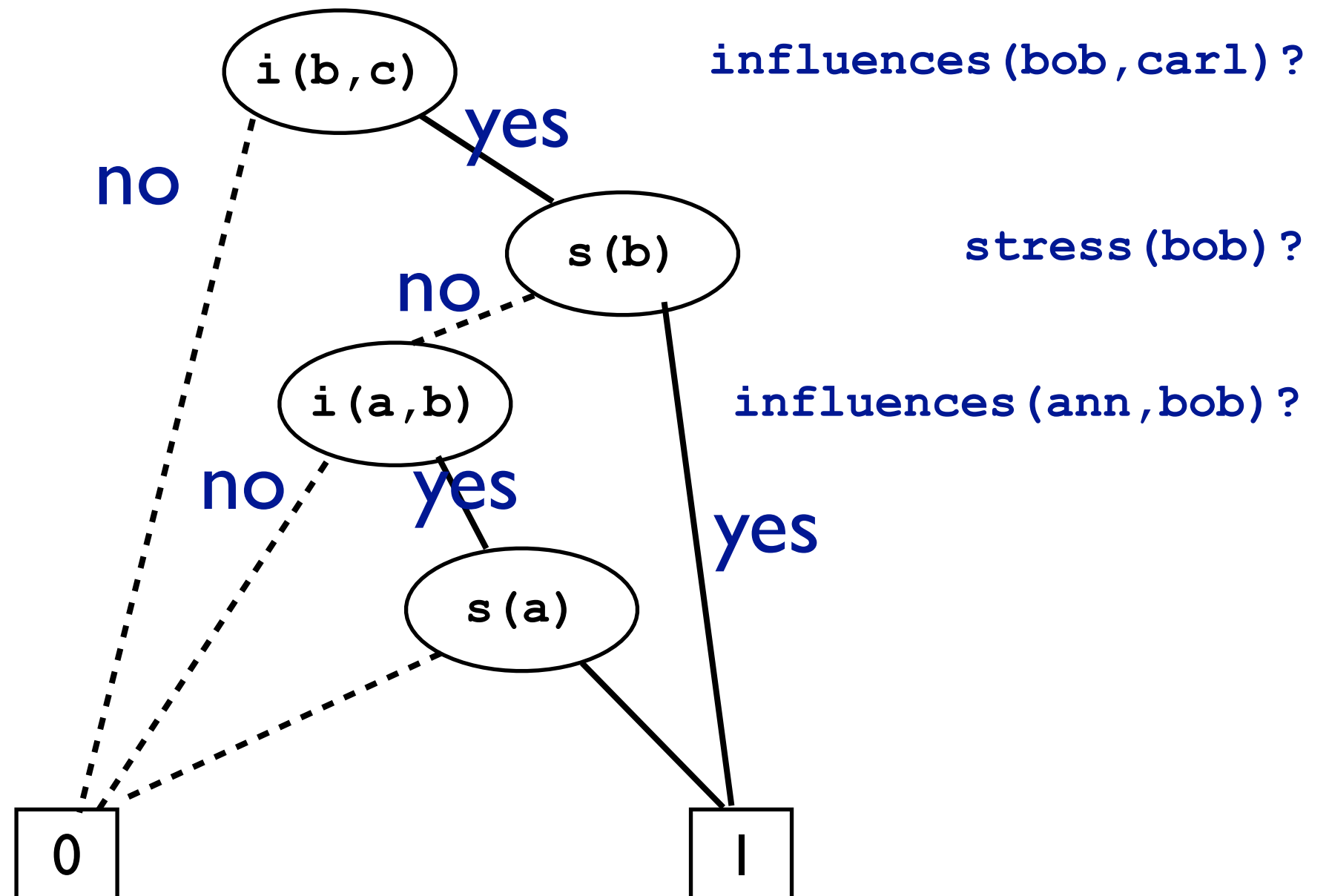
Binary Decision Diagrams



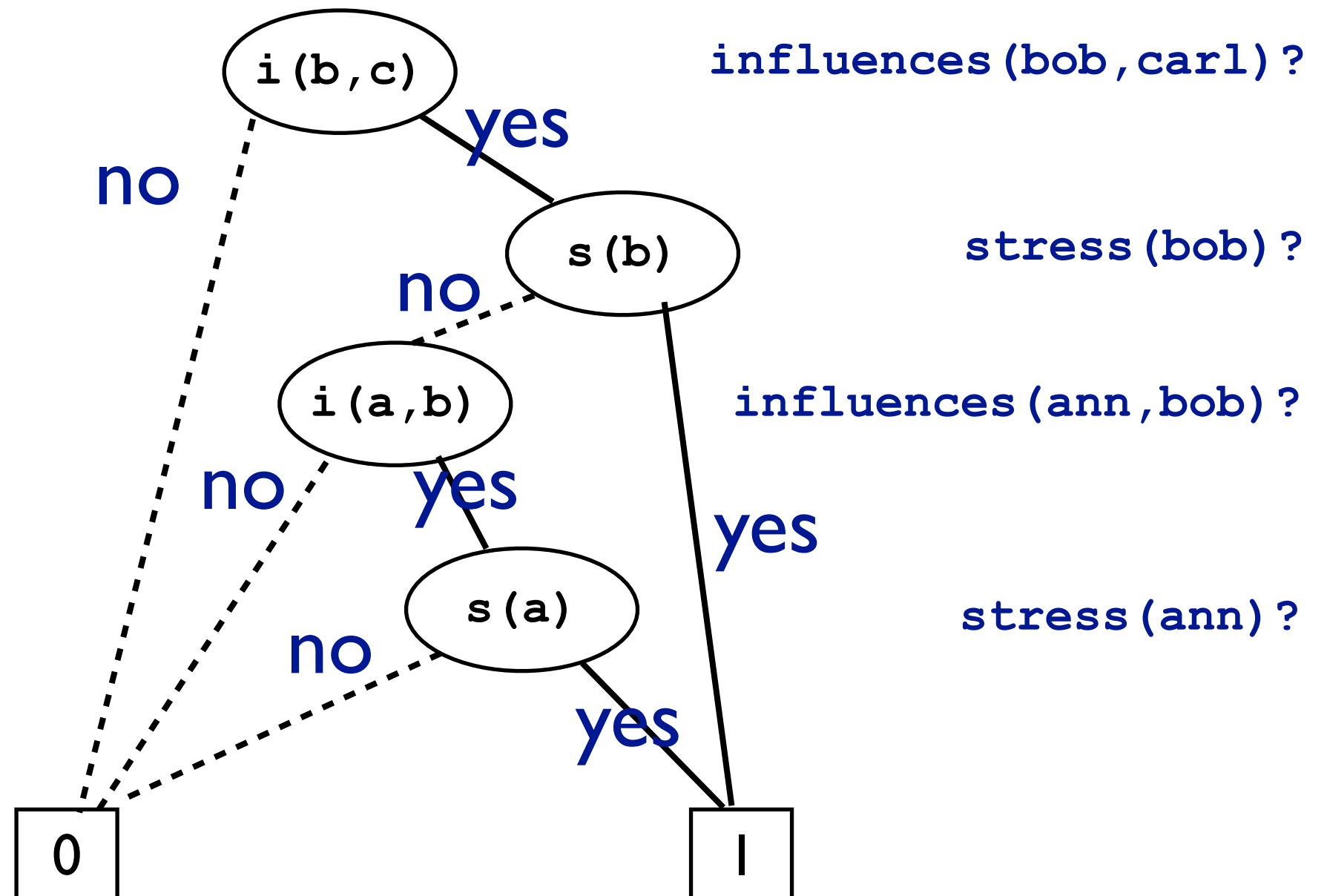
Binary Decision Diagrams



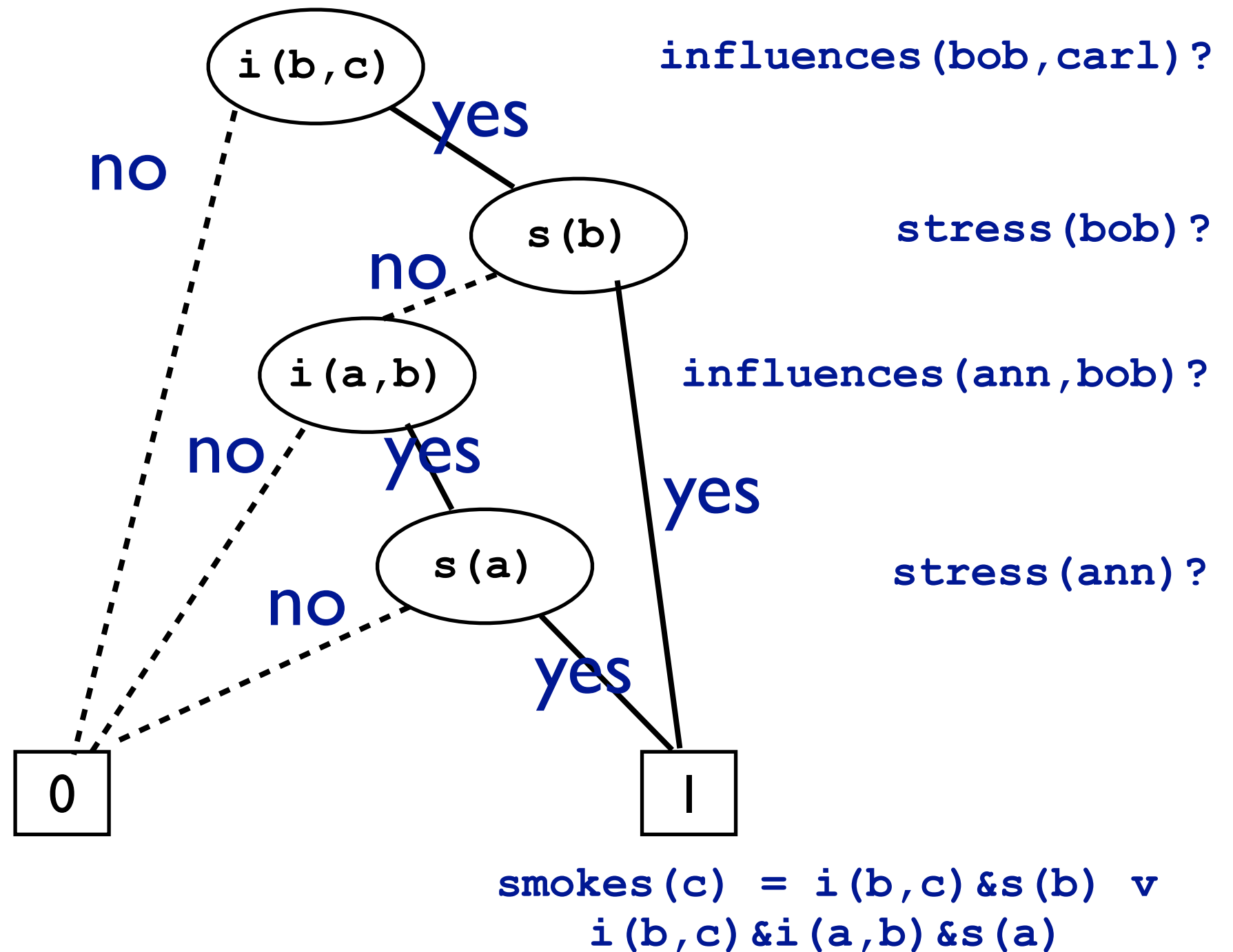
Binary Decision Diagrams



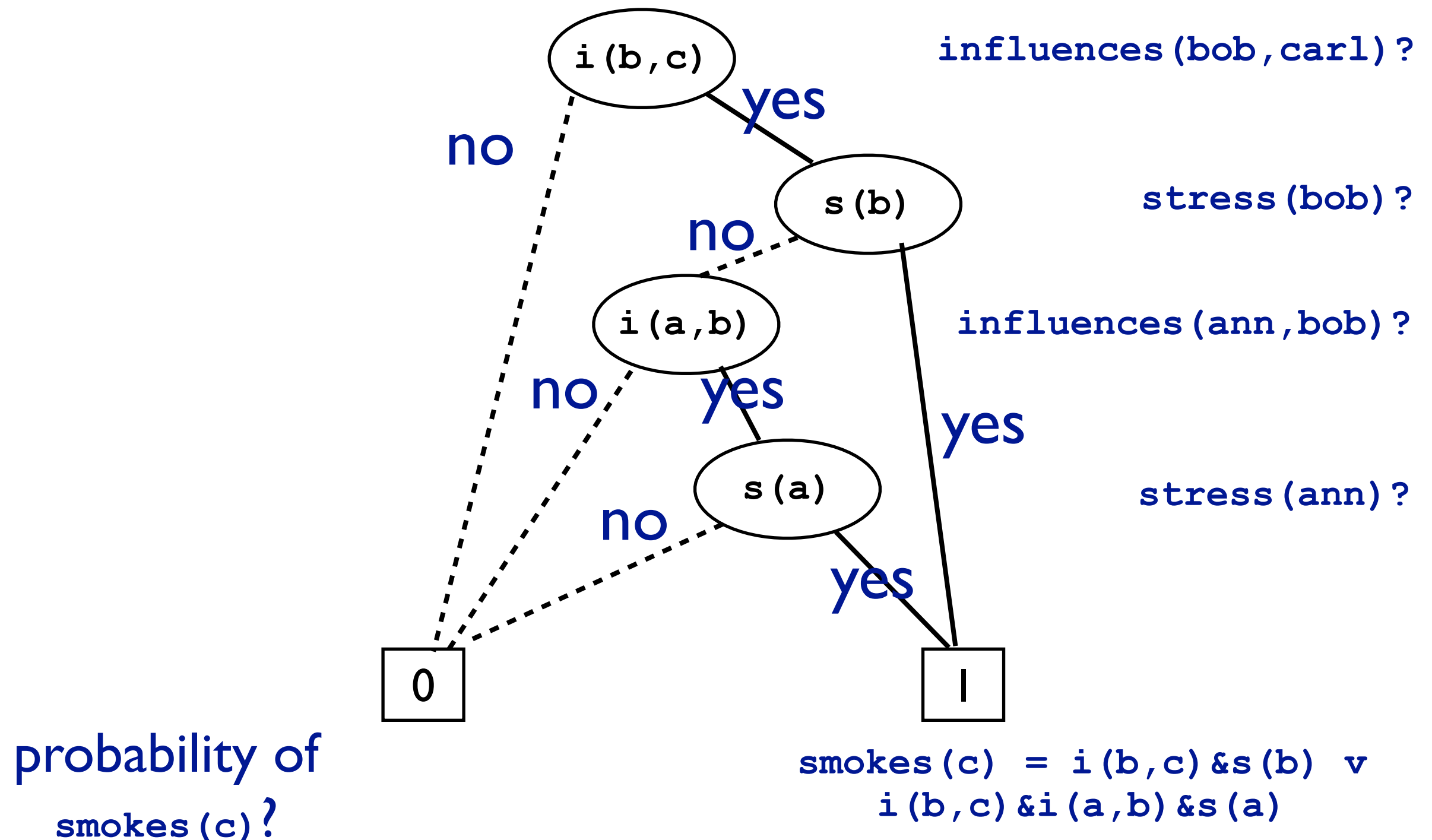
Binary Decision Diagrams



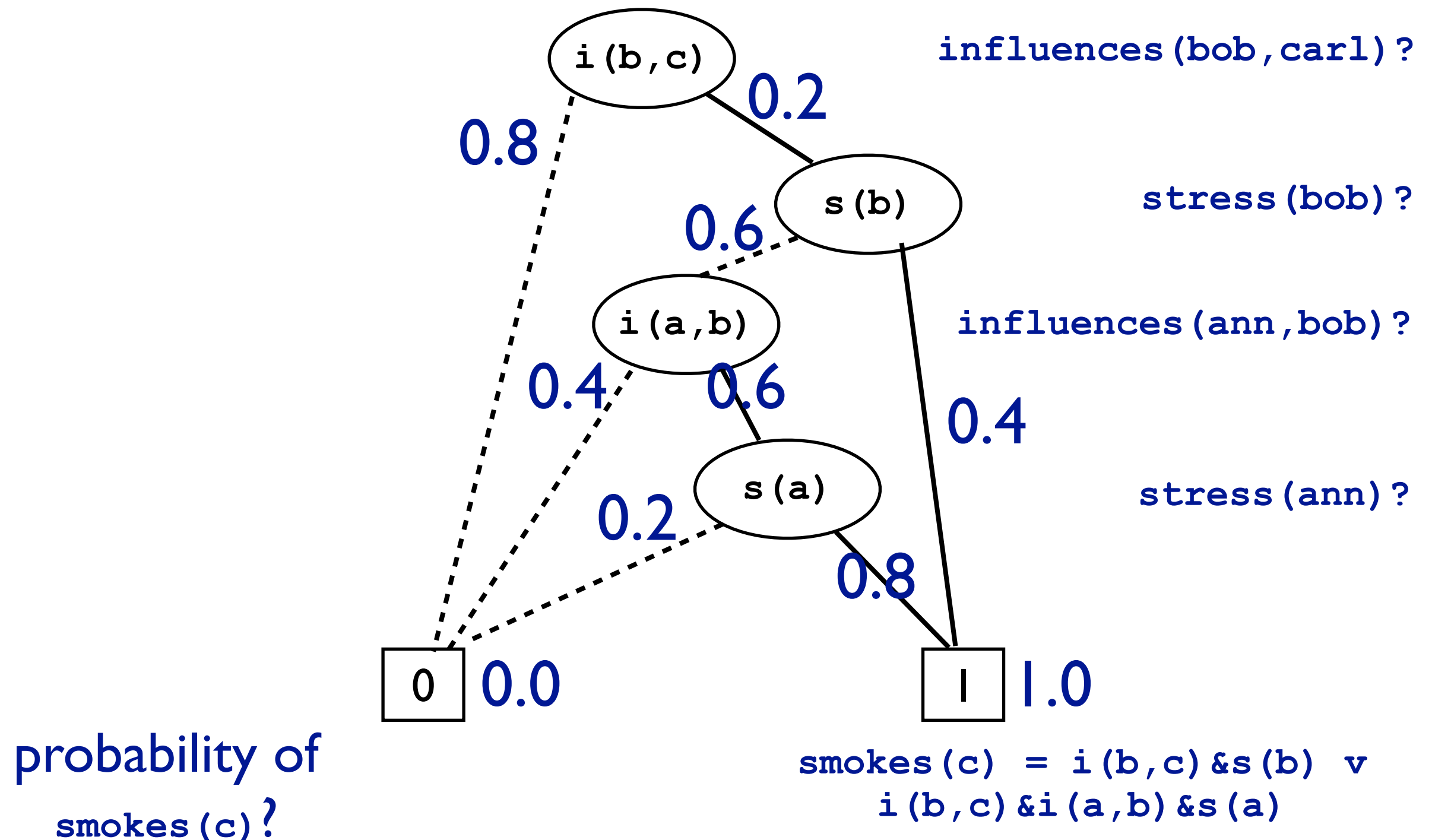
Binary Decision Diagrams



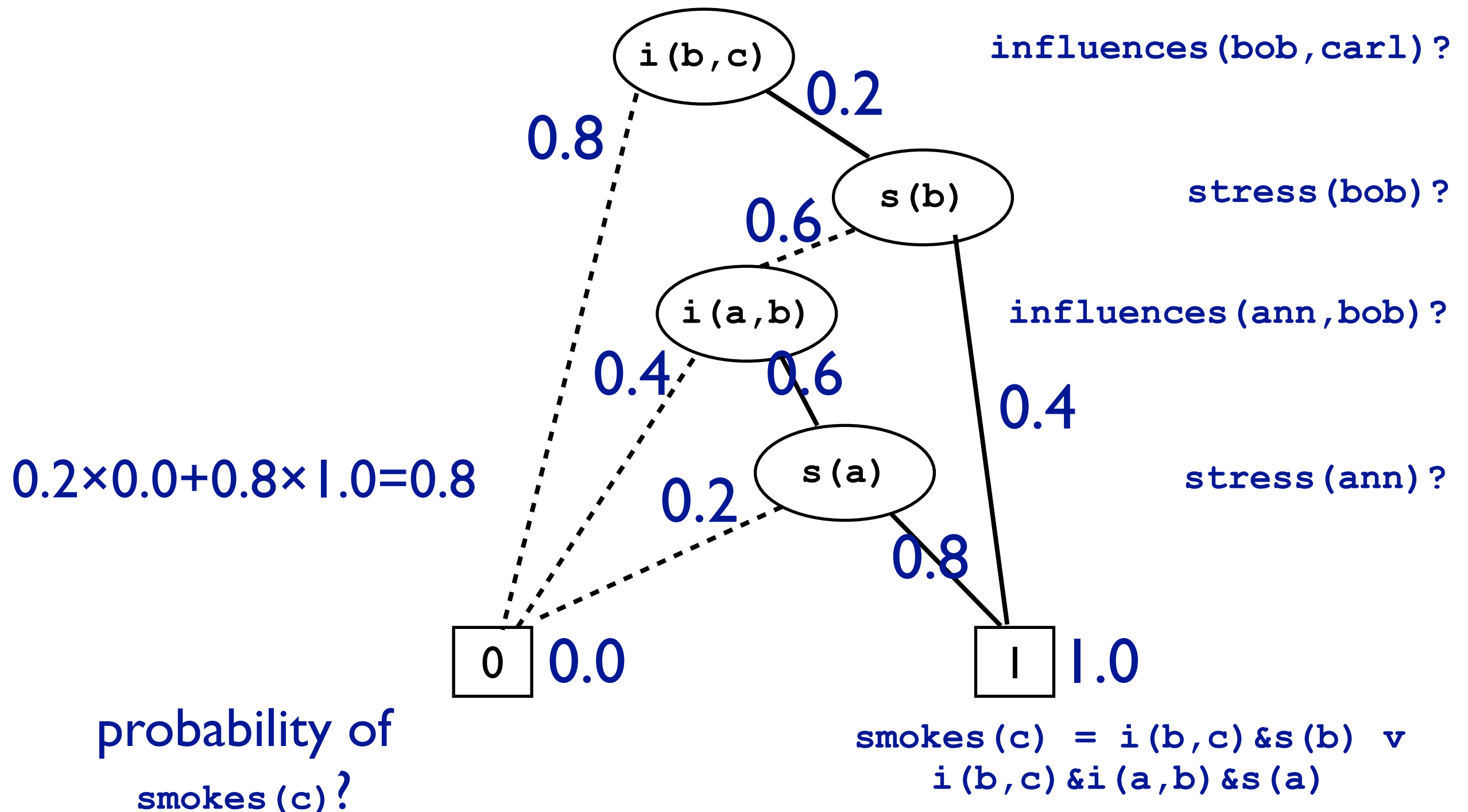
Binary Decision Diagrams



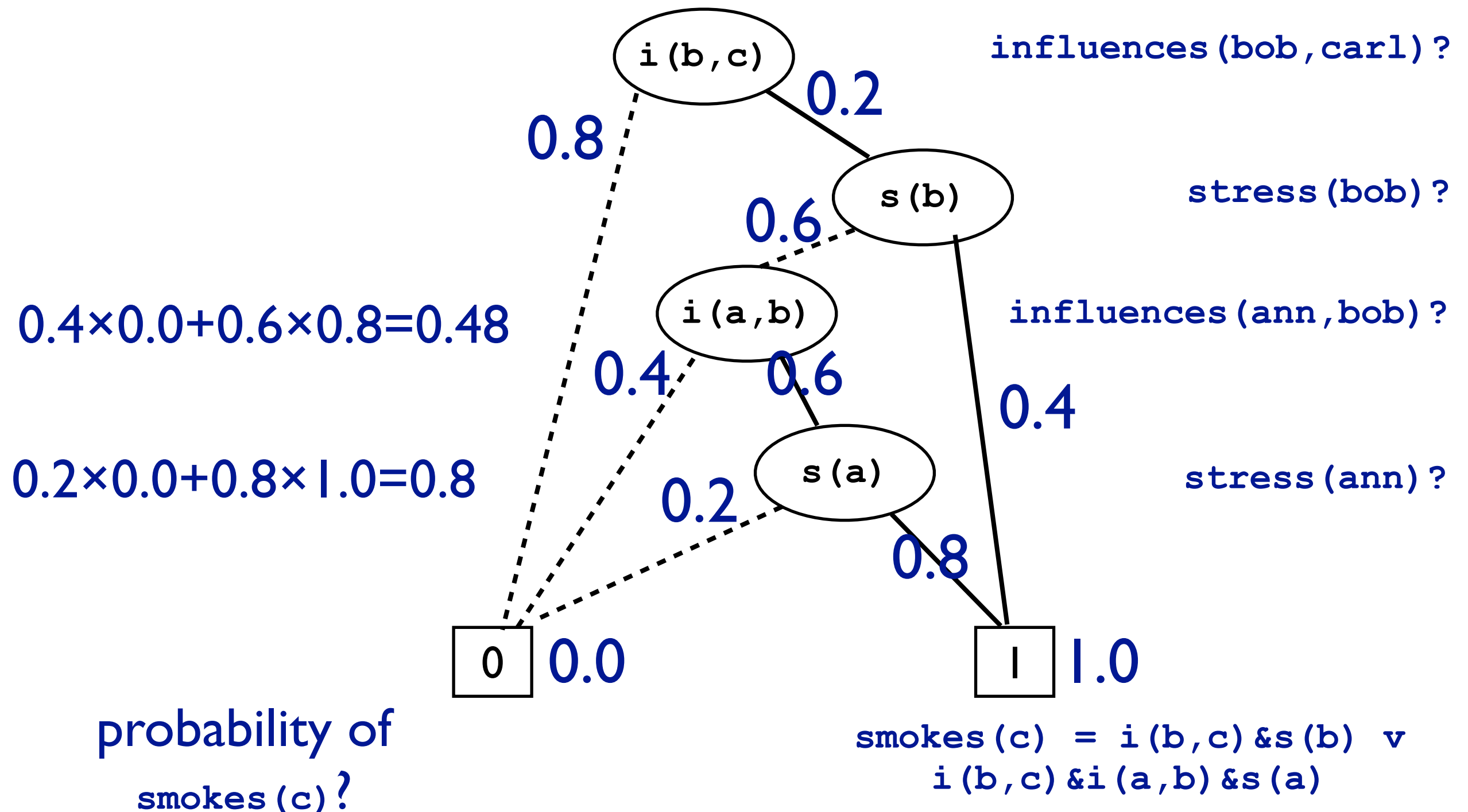
Binary Decision Diagrams



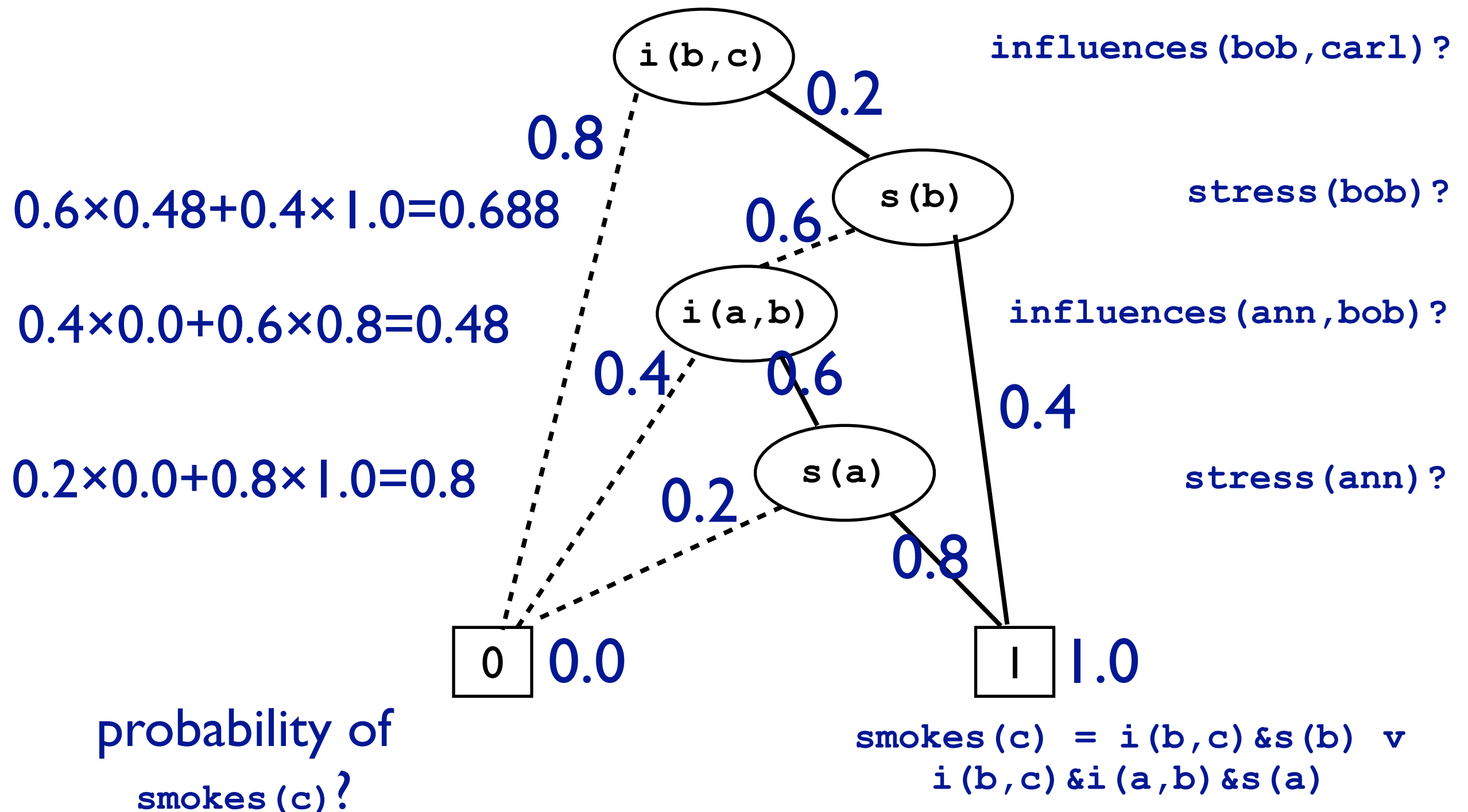
Binary Decision Diagrams



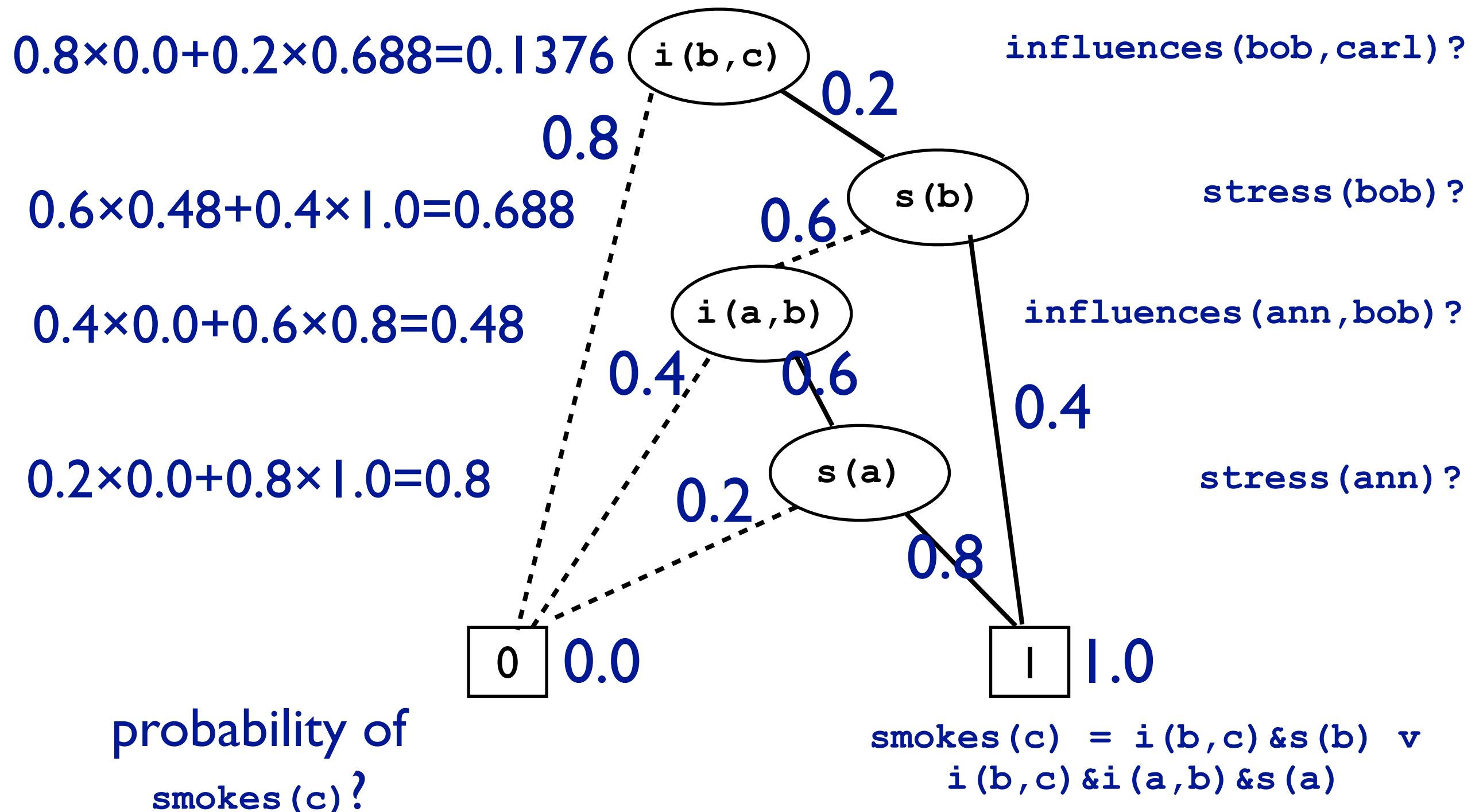
Binary Decision Diagrams



Binary Decision Diagrams



Binary Decision Diagrams



Initial Approach

(ProbLogI & others)

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

Initial Approach

(ProbLogI & others)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),heads(3).
```

win

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

Initial Approach

(ProbLogI & others)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),heads(3).
```

win

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

```
heads(1)  
heads(2) & heads(3)
```

Initial Approach

(ProbLogI & others)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),heads(3).
```

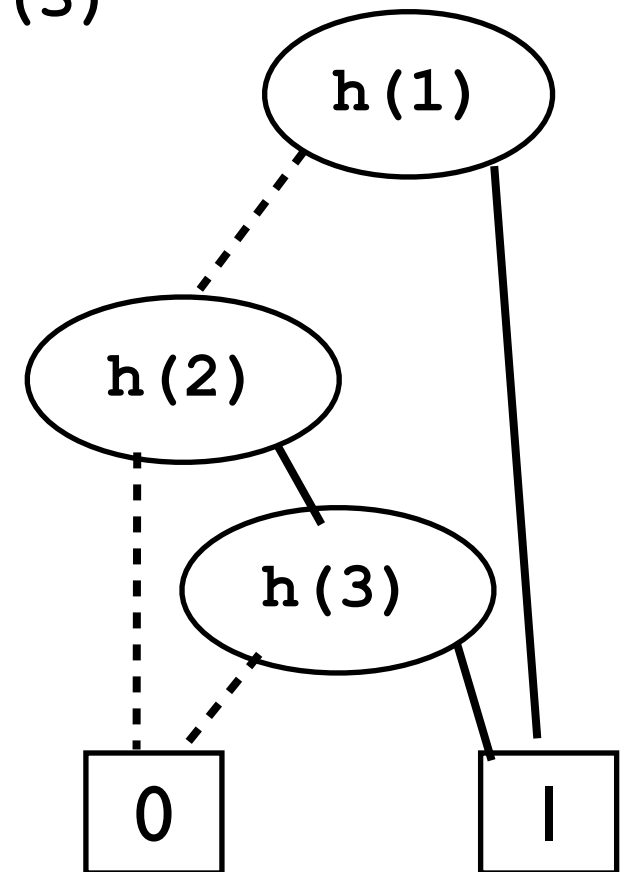
win

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

heads(1)
heads(2) & heads(3)



Initial Approach

(ProbLogI & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```

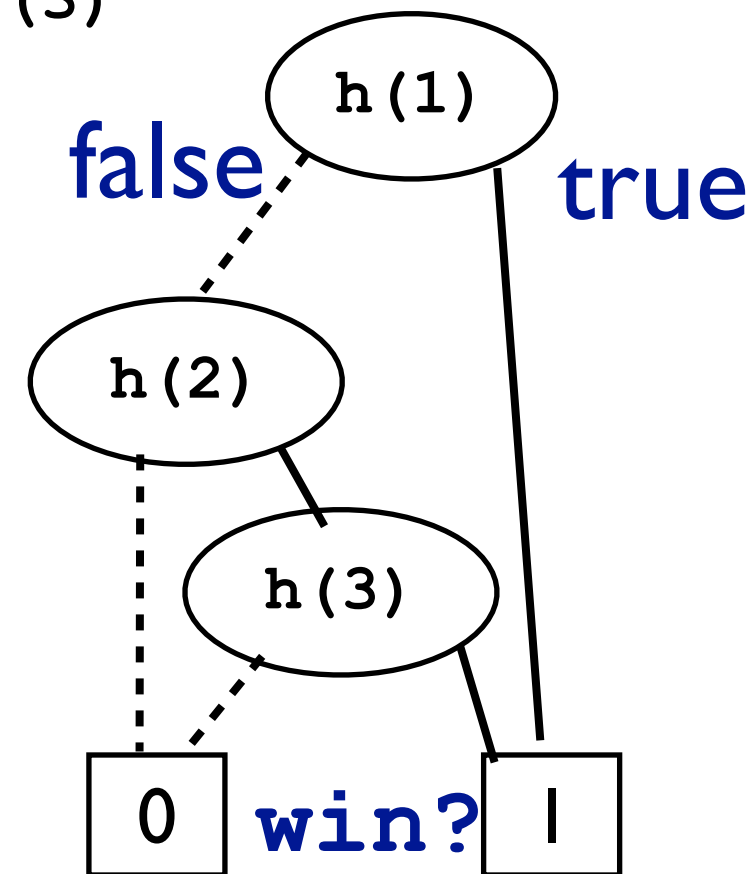
win

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

heads(1)
heads(2) & heads(3)



Initial Approach

(ProbLogI & others)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),heads(3).
```

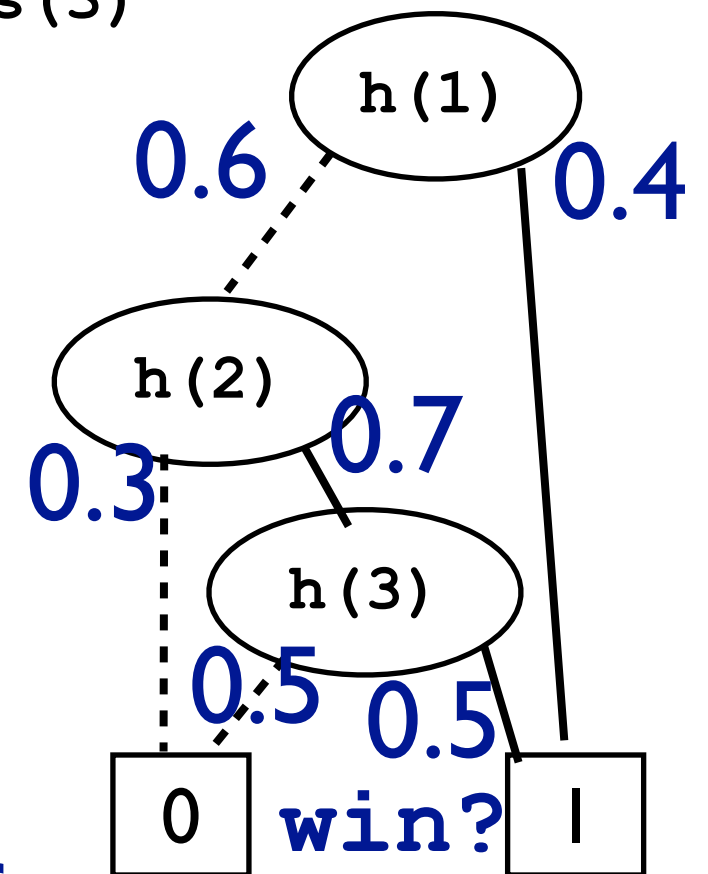
win

Find all proofs of query

Binary Decision
Diagram (BDD)

calculate marginal by
dynamic programming

heads(1)
heads(2) & heads(3)



$P(\text{win}) =$
probability of
reaching 1-leaf

Answering Questions

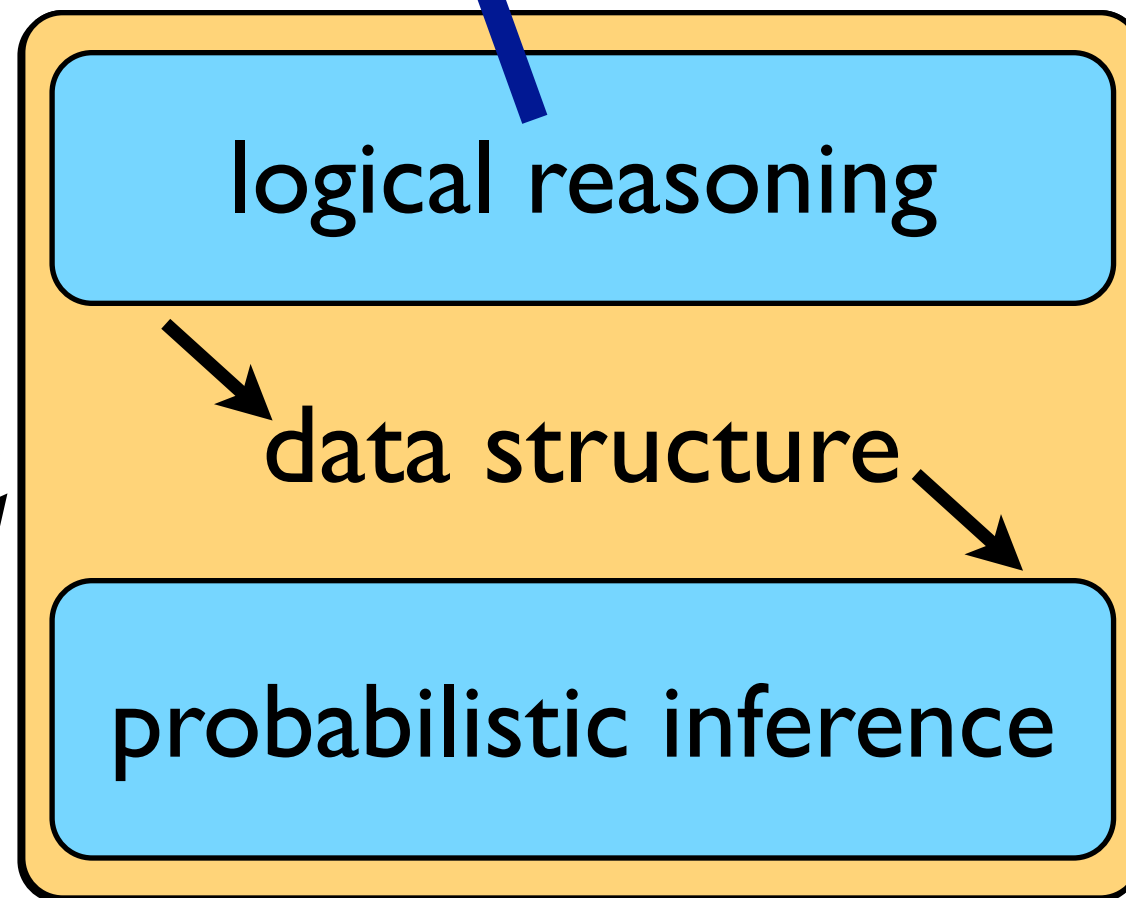
1. using proofs
2. using models

Given:

program

queries

evidence



Find:

marginal
probabilities

conditional
probabilities

MPE state

Logical Reasoning: Models in Prolog

```
?- smokes(carl) .
```

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

- Forward reasoning to construct unique model:

Logical Reasoning: Models in Prolog

```
?- smokes(carl) .
```

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

- Forward reasoning to construct unique model:

- Start with database facts

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

Logical Reasoning: Models in Prolog

```
?- smokes(carl) .
```

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

- Forward reasoning to construct unique model:

- Start with database facts
- Use rules to add more facts

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .  
  
smokes(ann) .
```

Logical Reasoning: Models in Prolog

```
?- smokes(carl) .
```

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

- Forward reasoning to construct unique model:
 - Start with database facts
 - Use rules to add more facts

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(ann) .  
smokes(bob) .
```

Logical Reasoning: Models in Prolog

```
?- smokes(carl) .
```

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

- Forward reasoning to construct unique model:
 - Start with database facts
 - Use rules to add more facts

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl)
```

```
smokes(ann) .  
smokes(bob) .  
smokes(carl) .
```

Logical Reasoning: Models in Prolog

```
?- smokes(carl) .
```

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

- Forward reasoning to construct unique model:
 - Start with database facts
 - Use rules to add more facts
- Query true iff in model

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(ann) .  
smokes(bob) .  
smokes(carl) .
```

Logical Reasoning: Models in Prolog

```
?- smokes(carl) .
```

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

- Forward reasoning to construct unique model:
 - Start with database facts
 - Use rules to add more facts
- Query true iff in model
- **ProbLog**: each possible world is a model, probability of query is sum over models where query is true

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(ann) .  
smokes(bob) .  
smokes(carl) .
```


Logical Reasoning: Models in Prolog

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

```
?- smokes(carl) .
```

- Forward reasoning to construct unique model:

- Start with database facts
- Use rules to add more facts

```
stress(ann) .  
influences(ann,bob) .  
influences(bob,carl) .
```

- Query true iff in model

```
smokes(ann) .  
smokes(bob) .  
smokes(carl) .
```

- **ProbLog**: each possible world is a model, probability of query is sum over models where query is true


→ weighted model counting

Weighted Model Counting

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$


Weighted Model Counting

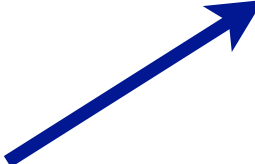
propositional formula in conjunctive normal form (CNF)


$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

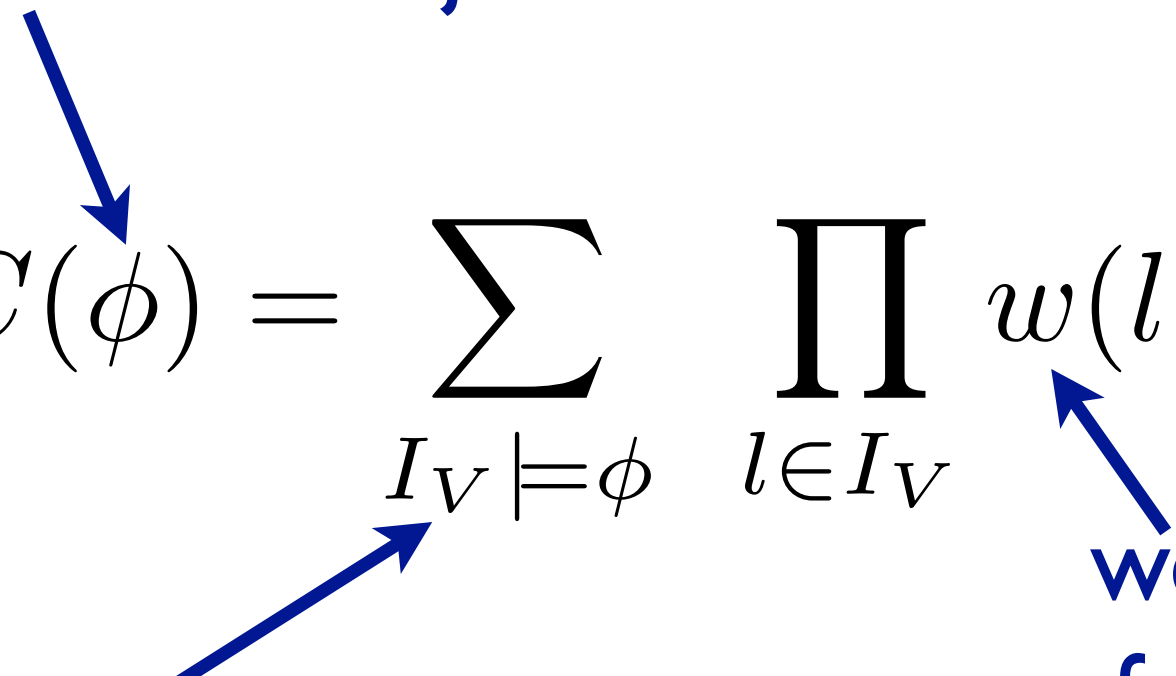

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$



interpretations (truth
value assignments) of
propositional variables

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)


$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

weight
of literal

interpretations (truth
value assignments) of
propositional variables

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)
given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

interpretations (truth
value assignments) of
propositional variables

weight
of literal

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)
given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

weight
of literal

interpretations (truth
value assignments) of
propositional variables
possible worlds

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)
given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

interpretations (truth
value assignments) of
propositional variables
possible worlds

weight
of literal
for $p::f$,
 $w(f) = p$
 $w(\text{not } f) = 1 - p$

Weighted

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

propositional formula in conjunctive normal form (CNF)
given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

interpretations (truth
value assignments) of
propositional variables
possible worlds

weight
of literal

for $p::f$,
 $w(f) = p$
 $w(\text{not } f) = 1 - p$

ProbLog \rightarrow CNF

?- smokes(carl) .

```
0.8::stress(ann) .  
0.4::stress(bob) .  
0.6::influences(ann,bob) .  
0.2::influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

ProbLog \rightarrow CNF

`?- smokes(carl) .`

```
0.8::stress(ann) .  
0.4::stress(bob) .  
0.6::influences(ann,bob) .  
0.2::influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

- Find relevant ground rules by backward reasoning

ProbLog \rightarrow CNF

?- smokes(carl) .

```
0.8::stress(ann) .  
0.4::stress(bob) .  
0.6::influences(ann,bob) .  
0.2::influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl) , smokes(bob) .  
smokes(bob)  :- stress(bob) .  
smokes(bob)  :- influences(ann,bob) , smokes(ann) .  
smokes(ann)  :- stress(ann) .
```

ProbLog \rightarrow CNF

`?- smokes(carl) .`

```
0.8::stress(ann) .
0.4::stress(bob) .
0.6::influences(ann,bob) .
0.2::influences(bob,carl) .
```

```
smokes(X) :- stress(X) .
smokes(X) :-
    influences(Y,X) ,
    smokes(Y) .
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl) , smokes(bob) .
smokes(bob)  :- stress(bob) .
smokes(bob)  :- influences(ann,bob) , smokes(ann) .
smokes(ann)  :- stress(ann) .
```

- Convert to propositional logic formula

ProbLog \rightarrow CNF

```
0.8::stress(ann).  
0.4::stress(bob).  
0.6::influences(ann,bob).  
0.2::influences(bob,carl).
```

```
?- smokes(carl).
```

```
smokes(X) :- stress(X).  
smokes(X) :-  
    influences(Y,X),  
    smokes(Y).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl), smokes(bob).  
smokes(bob) :- stress(bob).  
smokes(bob) :- influences(ann,bob), smokes(ann).  
smokes(ann) :- stress(ann).
```

- Convert to propositional logic formula

may require
loop-breaking

$$\begin{aligned} & \text{sm}(c) \leftrightarrow (\text{i}(b,c) \wedge \text{sm}(b)) \\ \wedge & \text{sm}(b) \leftrightarrow (\text{st}(b) \vee (\text{i}(a,b) \wedge \text{sm}(a))) \\ & \wedge \text{sm}(a) \leftrightarrow \text{st}(a) \end{aligned}$$

ProbLog \rightarrow CNF

```
?- smokes(carl) .
```

```
0.8::stress(ann) .  
0.4::stress(bob) .  
0.6::influences(ann,bob) .  
0.2::influences(bob,carl) .
```

```
smokes(X) :- stress(X) .  
smokes(X) :-  
    influences(Y,X) ,  
    smokes(Y) .
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl) , smokes(bob) .  
smokes(bob) :- stress(bob) .  
smokes(bob) :- influences(ann,bob) , smokes(ann) .  
smokes(ann) :- stress(ann) .
```

- Convert to propositional logic formula

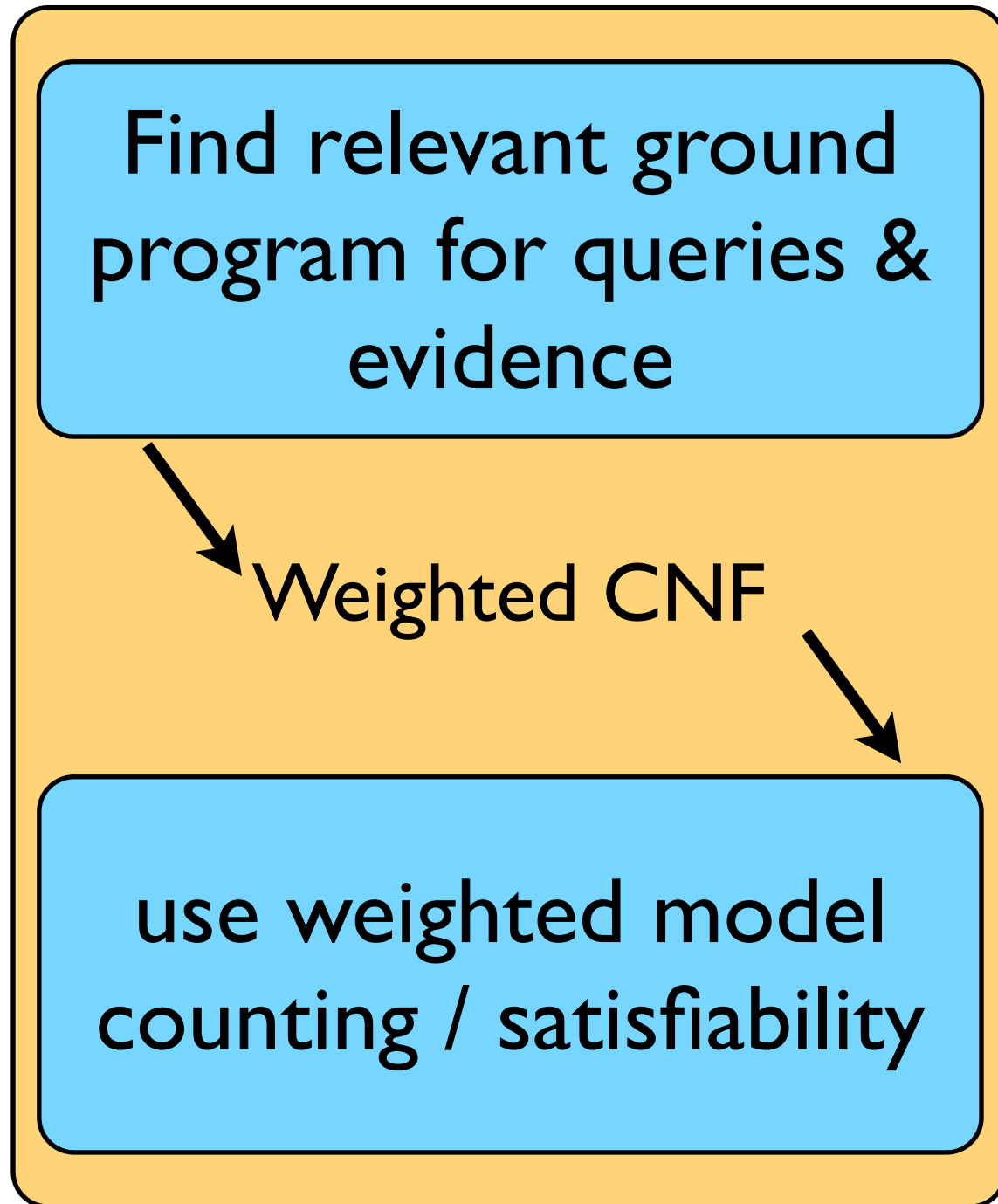
may require
loop-breaking

$$\begin{aligned} & \text{sm}(c) \leftrightarrow (\text{i}(b,c) \wedge \text{sm}(b)) \\ & \wedge \text{sm}(b) \leftrightarrow (\text{st}(b) \vee (\text{i}(a,b) \wedge \text{sm}(a))) \\ & \wedge \text{sm}(a) \leftrightarrow \text{st}(a) \end{aligned}$$

- Rewrite in CNF (as usual)

Current Approach

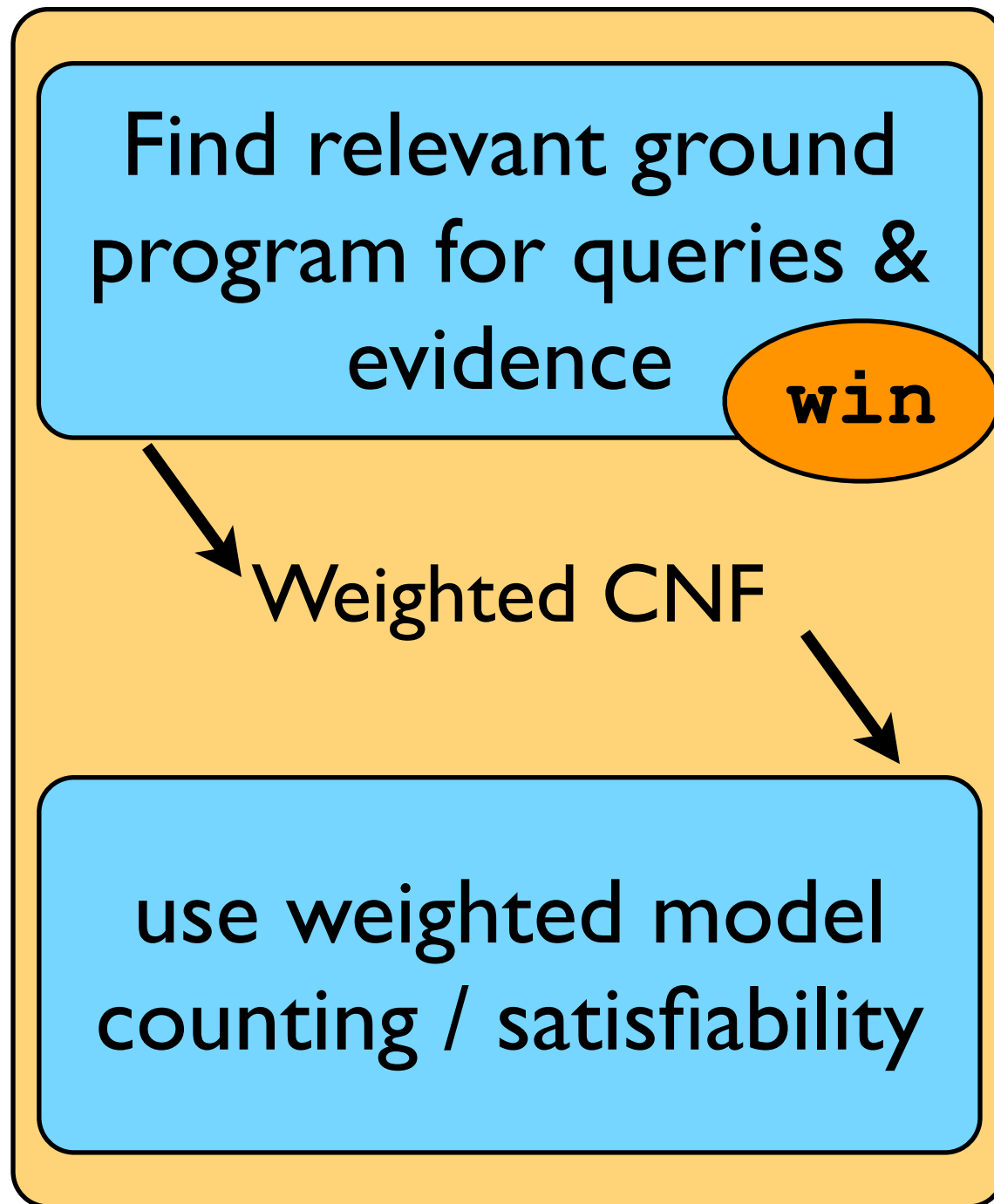
(ProbLog2)



Current Approach

(ProbLog2)

```
0.4::heads(1) .  
0.7::heads(2) .  
0.5::heads(3) .  
win :- heads(1) .  
win :- heads(2) ,  
           heads(3) .
```



Current Approach

(ProbLog2)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),  
        heads(3).
```

```
win :- heads(1).  
win :- heads(2), heads(3).
```

Find relevant ground
program for queries &
evidence

win

Weighted CNF

use weighted model
counting / satisfiability

Current Approach

(ProbLog2)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),  
        heads(3).
```

Find relevant ground
program for queries &
evidence

win

Weighted CNF

use weighted model
counting / satisfiability

```
win :- heads(1).  
win :- heads(2), heads(3).
```

↓
 $\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$

Current Approach

(ProbLog2)

```
0.4::heads(1).  
0.7::heads(2).  
0.5::heads(3).  
win :- heads(1).  
win :- heads(2),  
      heads(3).
```

Find relevant ground
program for queries &
evidence

win

Weighted CNF

use weighted model
counting / satisfiability

```
win :- heads(1).  
win :- heads(2), heads(3).
```

$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$

$(\neg \text{win} \vee h(1) \vee h(2))$
 $\wedge (\neg \text{win} \vee h(1) \vee h(3))$
 $\wedge (\text{win} \vee \neg h(1))$
 $\wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
      heads(3).
```

Find relevant ground
program for queries &
evidence

win

Weighted CNF

use weighted model
counting / satisfiability

```
win :- heads(1).
win :- heads(2), heads(3).
```

$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$

$(\neg \text{win} \vee h(1) \vee h(2))$
 $\wedge (\neg \text{win} \vee h(1) \vee h(3))$
 $\wedge (\text{win} \vee \neg h(1))$
 $\wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$

$h(1) \rightarrow 0.4$	$h(2) \rightarrow 0.7$	$h(3) \rightarrow 0.5$
$\neg h(1) \rightarrow 0.6$	$\neg h(2) \rightarrow 0.3$	$\neg h(3) \rightarrow 0.5$

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
      heads(3).
```

Find relevant ground
program for queries &
evidence

win

Weighted CNF

use weighted model
counting / satisfiability

```
win :- heads(1).
win :- heads(2), heads(3).
```

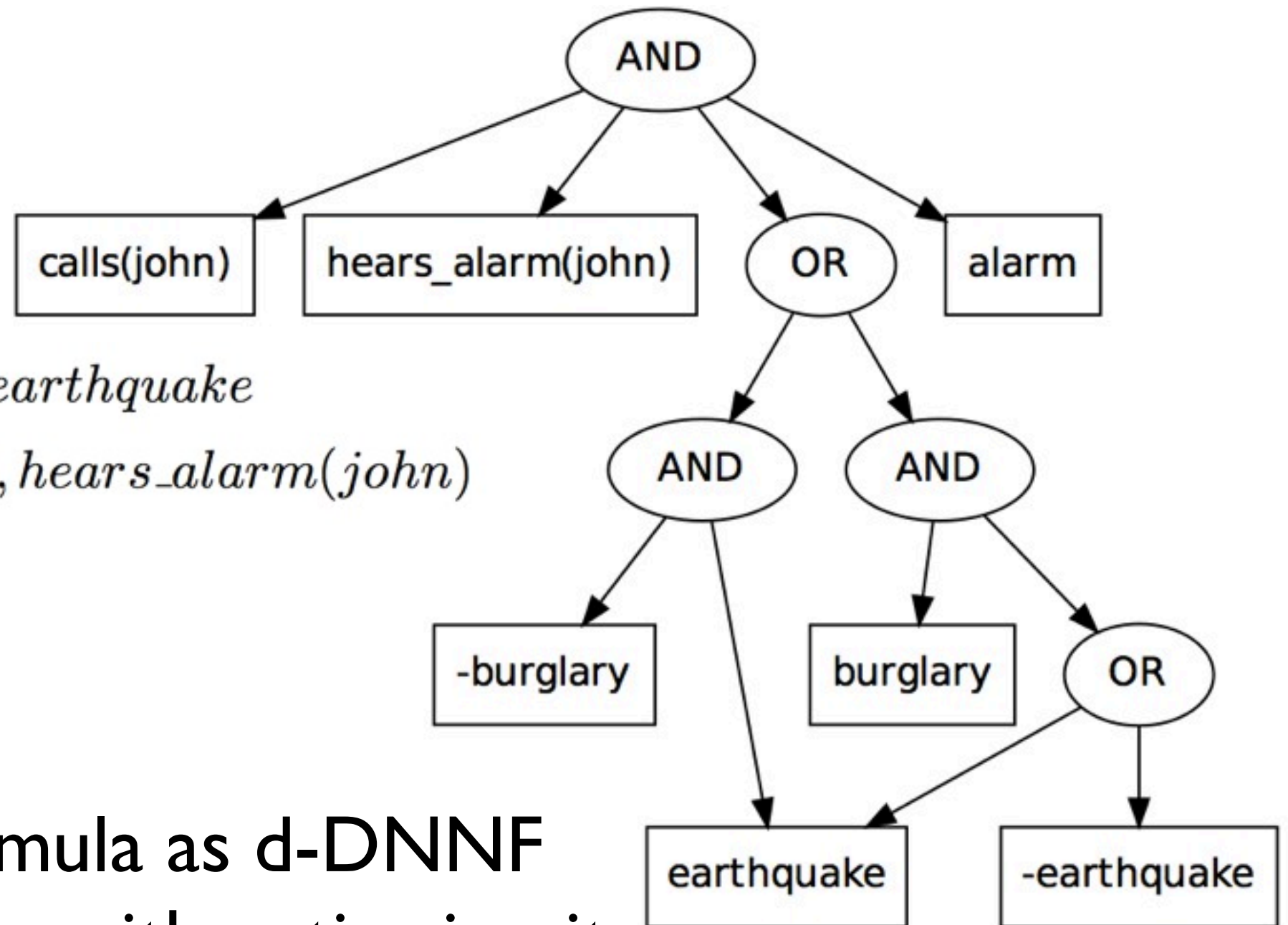
$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$

$(\neg \text{win} \vee h(1) \vee h(2))$
 $\wedge (\neg \text{win} \vee h(1) \vee h(3))$
 $\wedge (\text{win} \vee \neg h(1))$
 $\wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$

use
standard
tool

$h(1) \rightarrow 0.4$	$h(2) \rightarrow 0.7$	$h(3) \rightarrow 0.5$
$\neg h(1) \rightarrow 0.6$	$\neg h(2) \rightarrow 0.3$	$\neg h(3) \rightarrow 0.5$

WMC using d-DNNFs



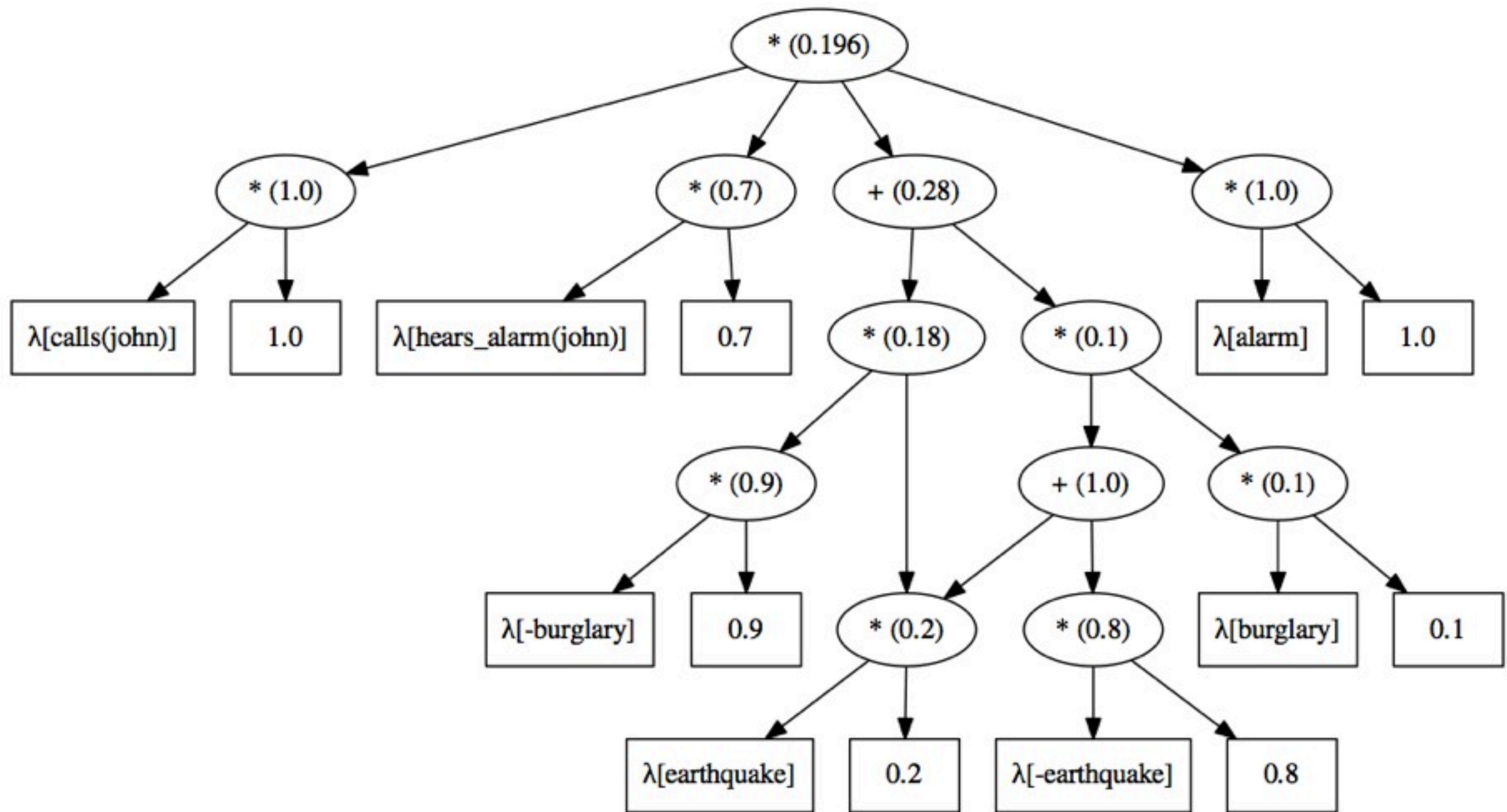
$alarm \leftrightarrow burglary \vee earthquake$

$calls(john) \leftrightarrow alarm, hears_alarm(john)$

$calls(john)$

1. represent formula as d-DNNF
2. transform into arithmetic circuit
3. evaluate bottom-up

WMC using d-DNNFs

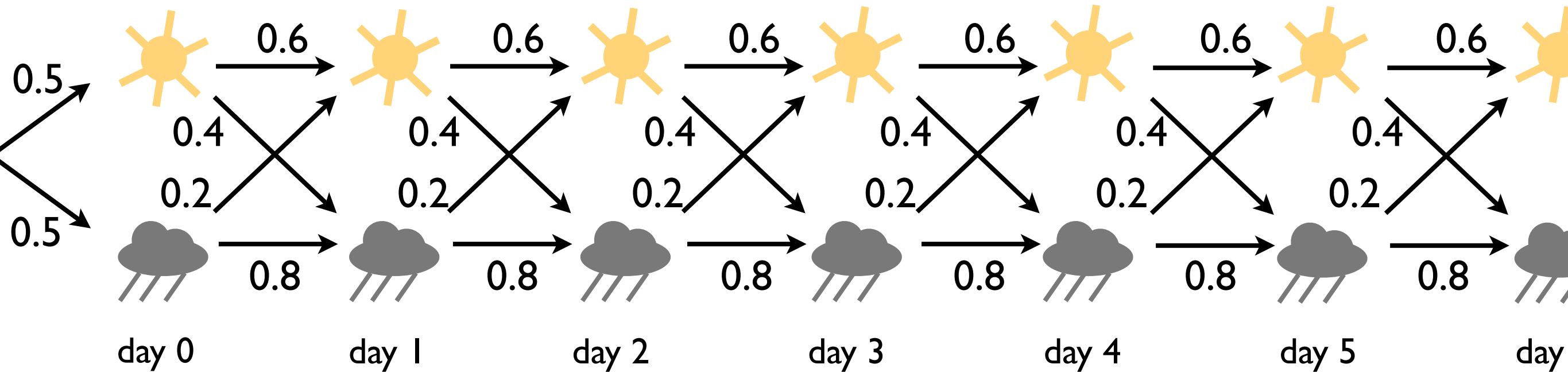


3. evaluate bottom-up

ProbLog Inference

- reduction to propositional formula
- addresses disjoint-sum-problem
- **but:** not all probabilistic logic programs face this problem! e.g., weather
- more generally: mutually exclusive proofs as assumed in PRISM

Weather in PRISM

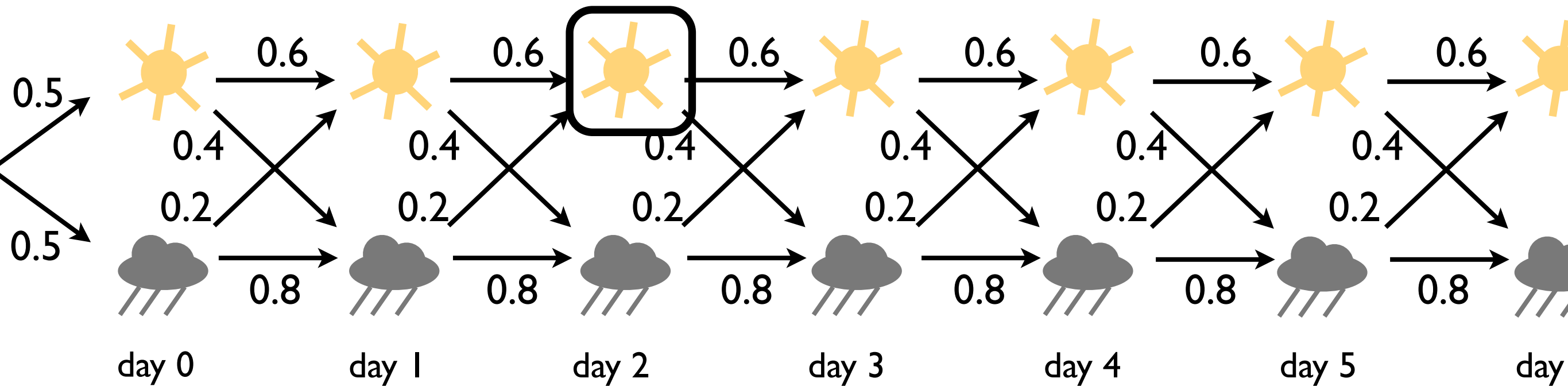


```
weather(W,Time) :-
    Time >= 0,
    msw(init,W0),
    w(0,Time,W0,W).
```

```
w(T,T,W,W).
```

```
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

Weather in PRISM

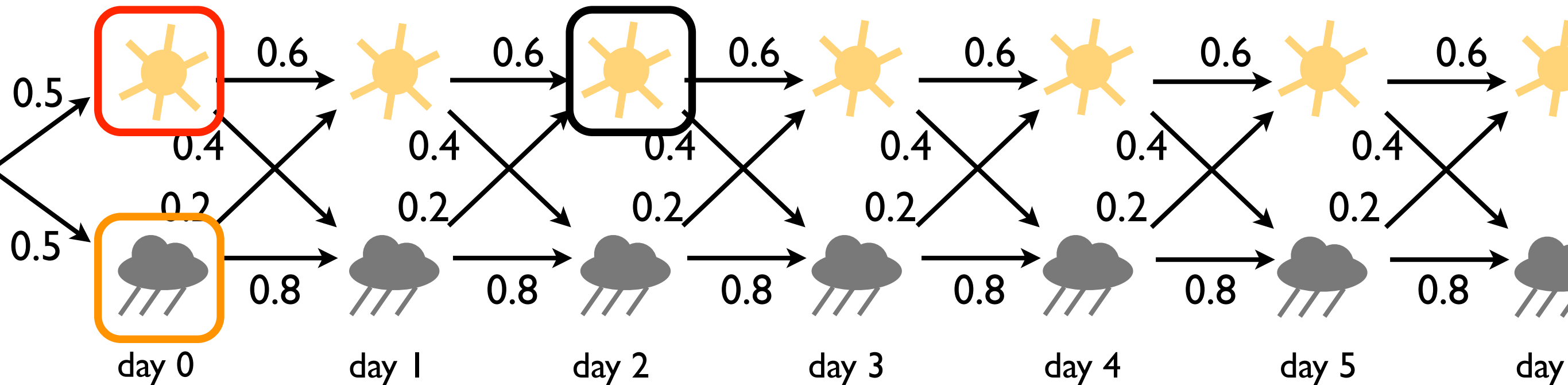


```
weather(W,Time) :-
    Time >= 0,
    msw(init,W0),
    w(0,Time,W0,W).
```

```
w(T,T,W,W).
```

```
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

Weather in PRISM



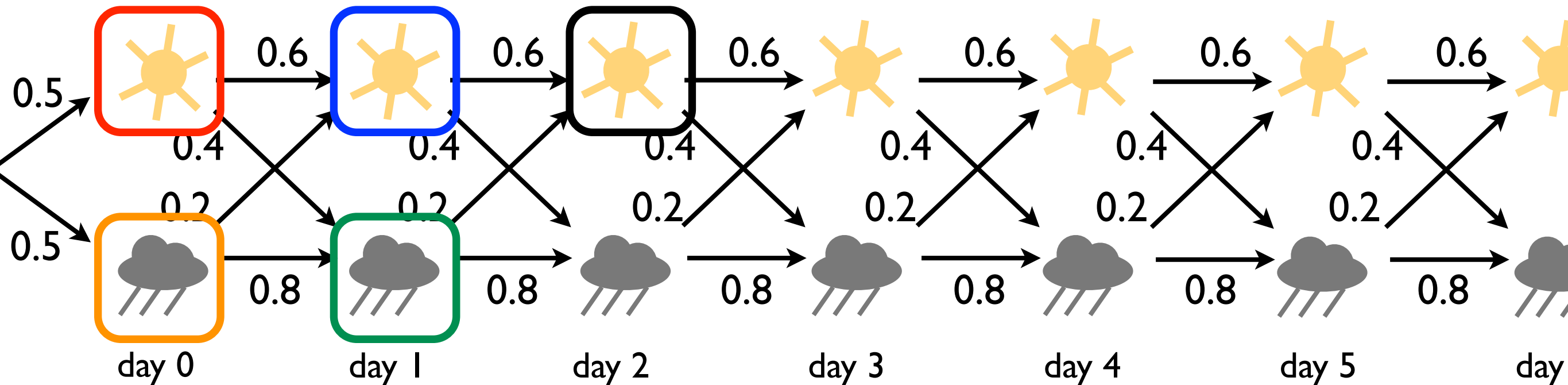
```
weather(W,Time) :-
    Time >= 0,
    msw(init,W0),
    w(0,Time,W0,W) .
```

```
weather(s,2) <=> msw(init,s) & w(0,2,s,s)
                v msw(init,r) & w(0,2,r,s)
```

```
w(T,T,W,W) .
```

```
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT) .
```

Weather in PRISM



```
weather(W,Time) :-
    Time >= 0,
    msw(init,W0),
    w(0,Time,W0,W).
```

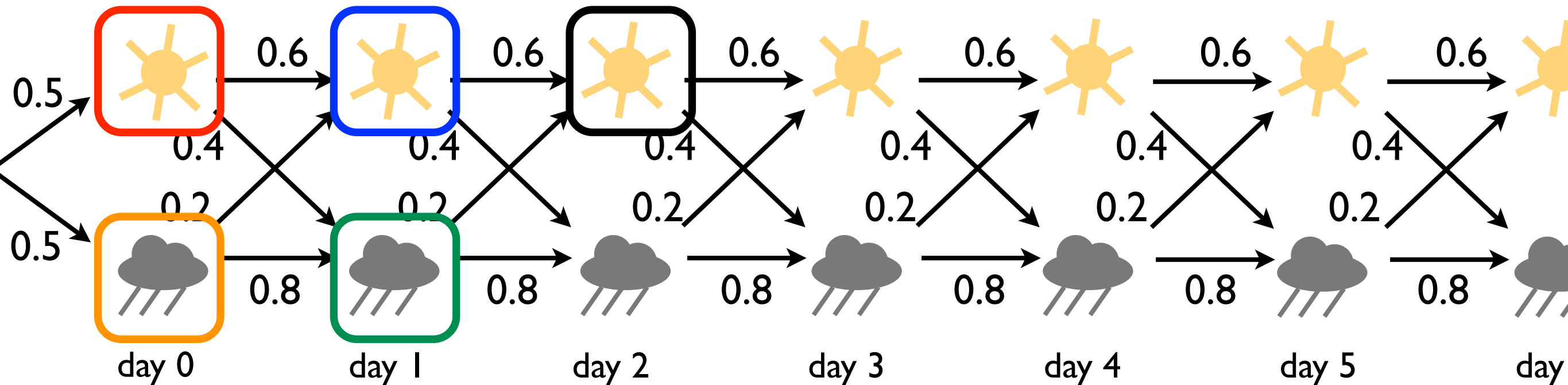
```
w(T,T,W,W).
```

```
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

```
weather(s,2) <=> msw(init,s) & w(0,2,s,s)
                  v msw(init,r) & w(0,2,r,s)
```

```
w(0,2,s,s) <=> msw(tr(s),s) & w(1,2,s,s)
                  v msw(tr(s),r) & w(1,2,r,s)
```

Weather in PRISM



```
weather(W,Time) :-
    Time >= 0,
    msw(init,W0),
    w(0,Time,W0,W).
```

```
w(T,T,W,W).
```

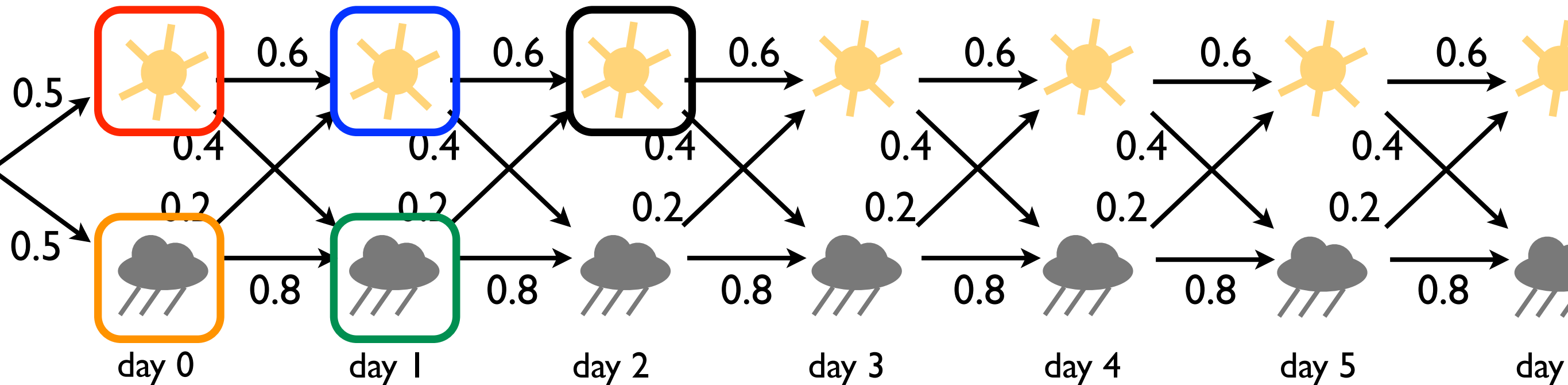
```
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

```
weather(s,2) <=> msw(init,s) & w(0,2,s,s)
                  v msw(init,r) & w(0,2,r,s)
```

```
w(0,2,s,s) <=> msw(tr(s),s) & w(1,2,s,s)
                  v msw(tr(s),r) & w(1,2,r,s)
```

```
w(0,2,r,s) <=> msw(tr(r),s) & w(1,2,r,s)
                  v msw(tr(r),r) & w(1,2,r,s)
```

Weather in PRISM



```
weather(W,Time) :-
    Time >= 0,
    msw(init,W0),
    w(0,Time,W0,W).
```

```
w(T,T,W,W).
```

```
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

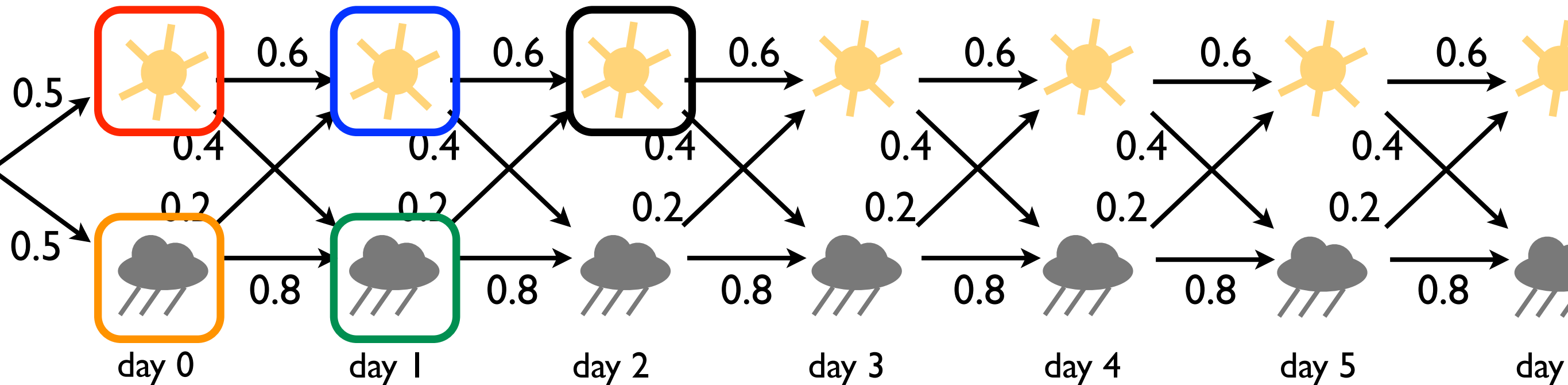
```
weather(s,2) <=> msw(init,s) & w(0,2,s,s)
                  v msw(init,r) & w(0,2,r,s)
```

```
w(0,2,s,s) <=> msw(tr(s),s) & w(1,2,s,s)
                  v msw(tr(s),r) & w(1,2,r,s)
```

```
w(0,2,r,s) <=> msw(tr(r),s) & w(1,2,r,s)
                  v msw(tr(r),r) & w(1,2,r,s)
```

```
w(1,2,s,s) <=> msw(tr(s),s)
```

Weather in PRISM



```

weather(W,Time) :-
    Time >= 0,
    msw(init,W0),
    w(0,Time,W0,W).

```

```

w(T,T,W,W).

```

```

w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).

```

```

weather(s,2) <=> msw(init,s) & w(0,2,s,s)
                  v msw(init,r) & w(0,2,r,s)

```

```

w(0,2,s,s) <=> msw(tr(s),s) & w(1,2,s,s)
                  v msw(tr(s),r) & w(1,2,r,s)

```

```

w(0,2,r,s) <=> msw(tr(r),s) & w(1,2,r,s)
                  v msw(tr(r),r) & w(1,2,r,s)

```

```

w(1,2,s,s) <=> msw(tr(s),s)

```

```

w(1,2,r,s) <=> msw(tr(r),s)

```


PRISM: compute probability by dynamic programming

`weather(s,2) <=> msw(init,s) & w(0,2,s,s) v msw(init,r) & w(0,2,r,s)`

`w(0,2,s,s) <=> msw(tr(s),s) & w(1,2,s,s) v msw(tr(s),r) & w(1,2,r,s)`

`w(0,2,r,s) <=> msw(tr(r),s) & w(1,2,r,s) v msw(tr(r),r) & w(1,2,r,s)`

`w(1,2,s,s) <=> msw(tr(s),s)`

`w(1,2,r,s) <=> msw(tr(r),s)`

PRISM: compute probability by dynamic programming

`weather(s,2) = msw(init,s) * w(0,2,s,s) + msw(init,r) * w(0,2,r,s)`

`w(0,2,s,s) = msw(tr(s),s) * w(1,2,s,s) + msw(tr(s),r) * w(1,2,r,s)`

`w(0,2,r,s) = msw(tr(r),s) * w(1,2,r,s) + msw(tr(r),r) * w(1,2,r,s)`

`w(1,2,s,s) = msw(tr(s),s)`

`w(1,2,r,s) = msw(tr(r),s)`

- replace
 - conjunction by multiplication
 - disjunction by summation

PRISM: compute probability by dynamic programming

`weather(s,2) = 0.5 * w(0,2,s,s) + 0.5 * w(0,2,r,s)`

`w(0,2,s,s) = 0.6 * w(1,2,s,s) + 0.4 * w(1,2,r,s)`

`w(0,2,r,s) = 0.2 * w(1,2,r,s) + 0.8 * w(1,2,r,s)`

`w(1,2,s,s) = 0.6`

`w(1,2,r,s) = 0.2`

- replace
 - conjunction by multiplication
 - disjunction by summation
 - msw-atoms by their probability

PRISM: compute probability by dynamic programming

$$\text{weather}(s, 2) = 0.5 * 0.44 + 0.5 * 0.28 = 0.36$$

$$w(0, 2, s, s) = 0.6 * 0.6 + 0.4 * 0.2 = 0.44$$

$$w(0, 2, r, s) = 0.2 * 0.6 + 0.8 * 0.2 = 0.28$$

$$w(1, 2, s, s) = 0.6$$

$$w(1, 2, r, s) = 0.2$$

- replace
 - conjunction by multiplication
 - disjunction by summation
 - msw-atoms by their probability
- propagate values

Dyna

[Eisner et al 05]

- Weighted Logic Programs (but not Prolog)
- Inspired by NLP
- Arbitrary semiring weights
- Forward reasoning

CYK parsing in Dyna

```
word(john,0,1)=1  
word(likes,1,2)=1  
word(mary,2,3)=1  
end(3)=1
```

input sentence

```
rewrite(np,mary)=0.003  
rewrite(vp,verb,np)=0.5  
...
```

probabilistic grammar rules

rules $X \rightarrow W$ with W spanning I,J



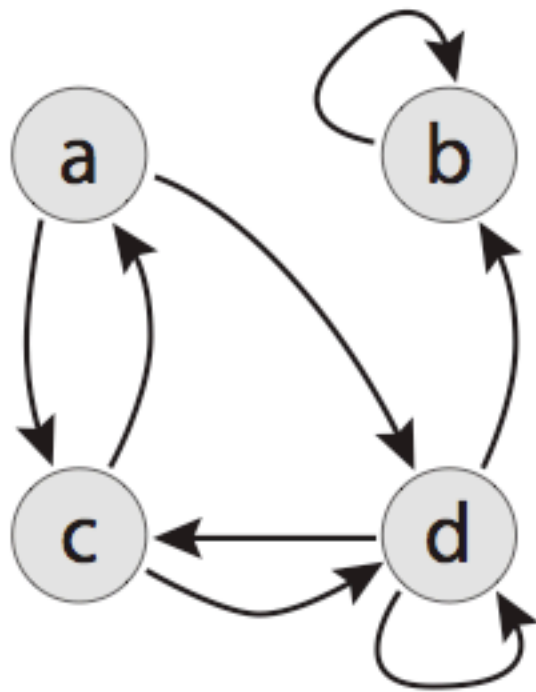
```
constit(X,I,J) += rewrite(X,W) * word(W,I,J).  
constit(X,I,K) += rewrite(X,Y,Z) * constit(Y,I,J) * constit(Z,J,K).  
goal += constit(s,0,N) * end(N).
```

rules $X \rightarrow YZ$, split points J

sum over all parses

```
reachable(Q) :- initial(Q).  
reachable(Q) :- reachable(P), edge(P, Q).
```

Fig. 1. A simple bottom-up logic program for graph reachability



<code>initial(a) = T</code>	<code>edge(c, d) = T</code>
<code>edge(a, c) = T</code>	<code>edge(d, b) = T</code>
<code>edge(a, d) = T</code>	<code>edge(d, c) = T</code>
<code>edge(b, b) = T</code>	<code>edge(d, d) = T</code>
<code>edge(c, a) = T</code>	

Fig. 2. A directed graph and the corresponding initial database

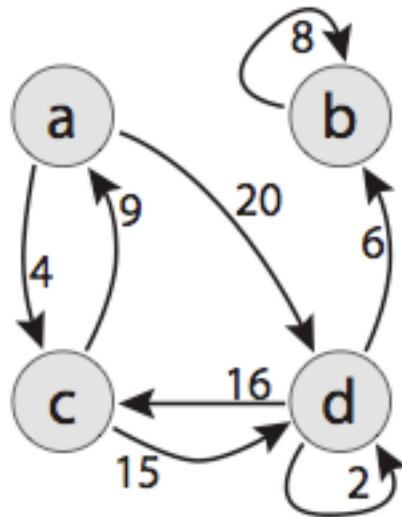
$\text{reachable}(Q) \oplus = \text{initial}(Q).$

$\text{reachable}(Q) \oplus = \text{reachable}(P) \otimes \text{edge}(P, Q).$

$\oplus \rightarrow \text{min}$

$\otimes \rightarrow +$

shortest
path

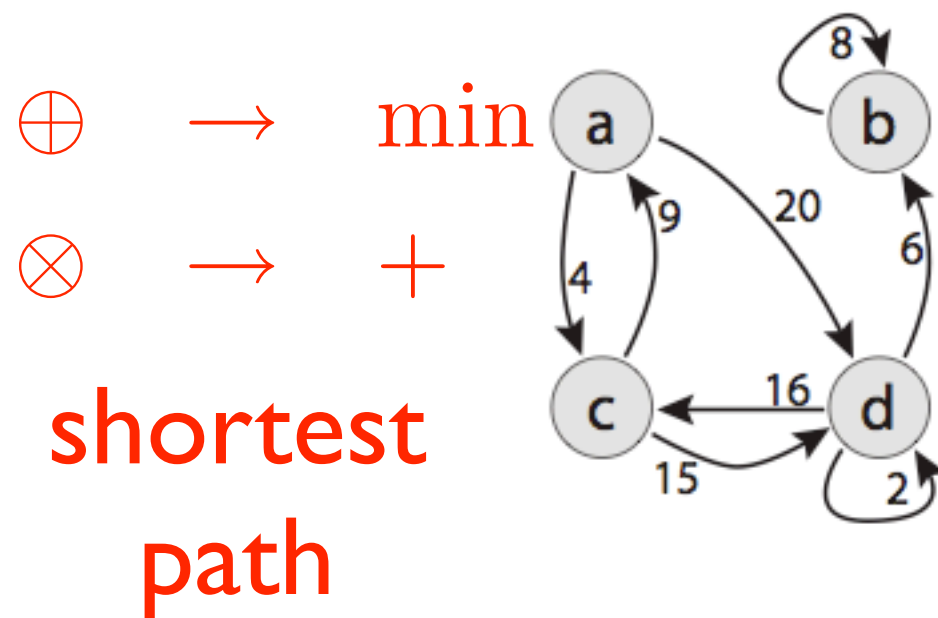


$\text{initial}(a) = 0$	$\text{edge}(c, d) = 15$
$\text{edge}(a, c) = 4$	$\text{edge}(d, b) = 6$
$\text{edge}(a, d) = 20$	$\text{edge}(d, c) = 16$
$\text{edge}(b, b) = 8$	$\text{edge}(d, d) = 2$
$\text{edge}(c, a) = 9$	

Fig. 3. A cost graph and the corresponding initial database

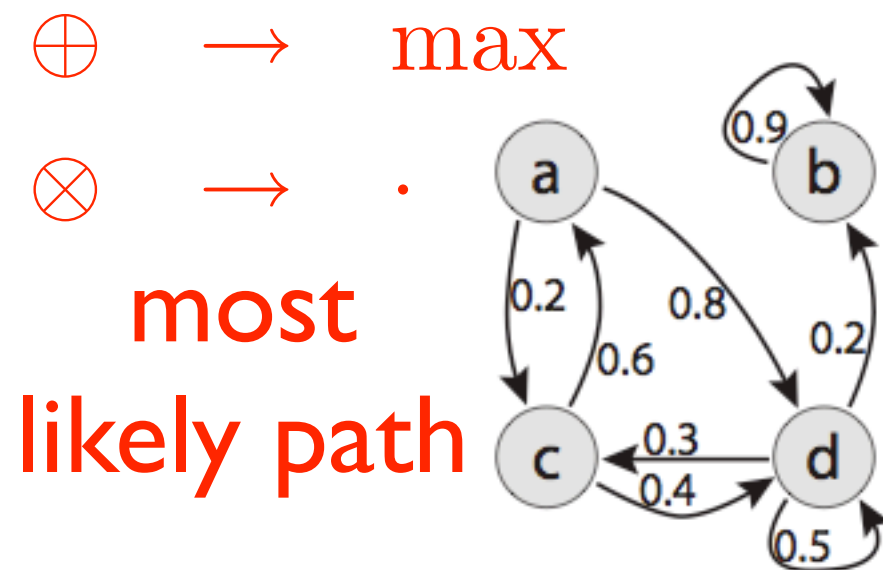
$\text{reachable}(Q) \oplus = \text{initial}(Q).$

$\text{reachable}(Q) \oplus = \text{reachable}(P) \otimes \text{edge}(P, Q).$



$\text{initial}(a) = 0$	$\text{edge}(c, d) = 15$
$\text{edge}(a, c) = 4$	$\text{edge}(d, b) = 6$
$\text{edge}(a, d) = 20$	$\text{edge}(d, c) = 16$
$\text{edge}(b, b) = 8$	$\text{edge}(d, d) = 2$
$\text{edge}(c, a) = 9$	

Fig. 3. A cost graph and the corresponding initial database



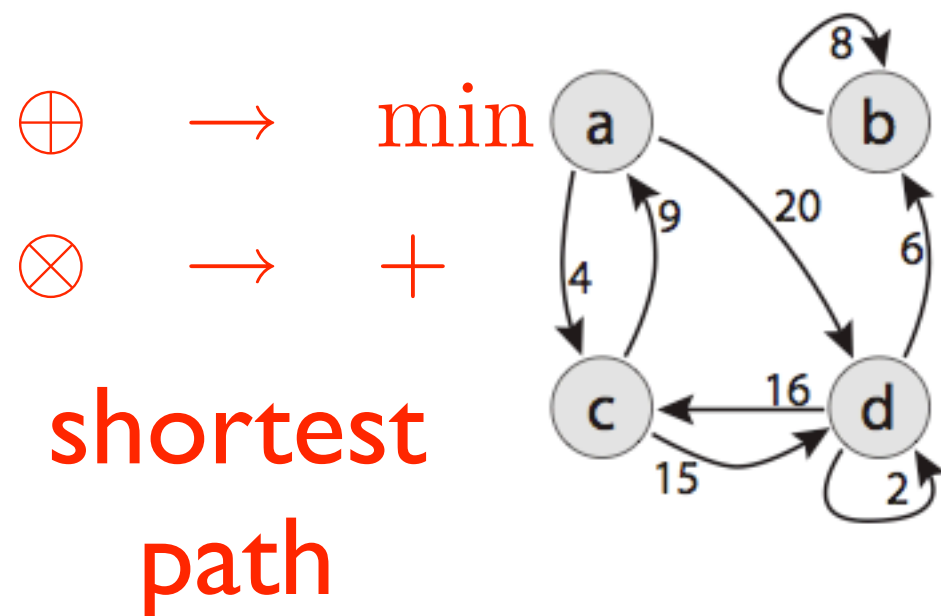
$\text{initial}(a) = 1$	$\text{edge}(c, d) = 0.4$
$\text{edge}(a, c) = 0.2$	$\text{edge}(d, b) = 0.2$
$\text{edge}(a, d) = 0.8$	$\text{edge}(d, c) = 0.3$
$\text{edge}(b, b) = 0.9$	$\text{edge}(d, d) = 0.5$
$\text{edge}(c, a) = 0.6$	

Fig. 4. A probabilistic graph and the corresponding initial database. With stopping probabilities made explicit, this would encode a Markov model.

[Figures: Cohen et al, ICLP 08]

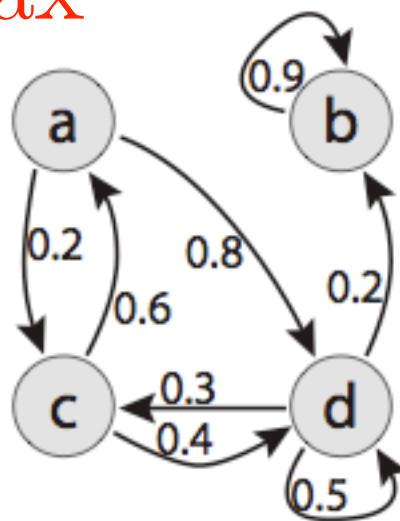
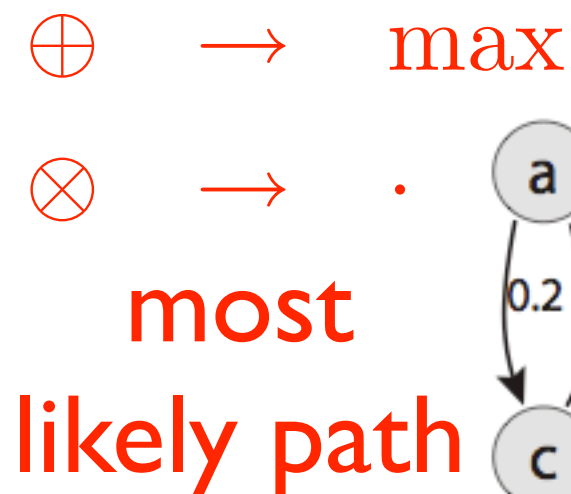
$\text{reachable}(Q) \oplus = \text{initial}(Q).$

$\text{reachable}(Q) \oplus = \text{reachable}(P) \otimes \text{edge}(P, Q).$



$\text{initial}(a) = 0$	$\text{edge}(c, d) = 15$
$\text{edge}(a, c) = 4$	$\text{edge}(d, b) = 6$
$\text{edge}(a, d) = 20$	$\text{edge}(d, c) = 16$
$\text{edge}(b, b) = 8$	$\text{edge}(d, d) = 2$
$\text{edge}(c, a) = 9$	

Fig. 3. A cost graph and the corresponding initial database



$\text{initial}(a) = 1$	$\text{edge}(c, d) = 0.4$
$\text{edge}(a, c) = 0.2$	$\text{edge}(d, b) = 0.2$
$\text{edge}(a, d) = 0.8$	$\text{edge}(d, c) = 0.3$
$\text{edge}(b, b) = 0.9$	$\text{edge}(d, d) = 0.5$
$\text{edge}(c, a) = 0.6$	

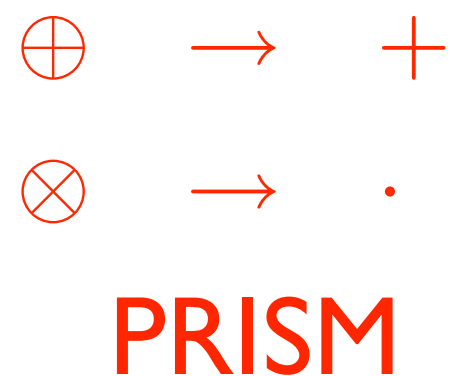


Fig. 4. A probabilistic graph and the corresponding initial database. With stopping probabilities made explicit, this would encode a Markov model.

[Figures: Cohen et al, ICLP 08]

Semantics

- Semiring *
- valuation function maps provable ground terms to values
- extended to expressions, e.g. $\llbracket x * y \rrbracket \stackrel{\text{def}}{=} \llbracket x \rrbracket * \llbracket y \rrbracket$
- weighted rules $r \oplus_r = E$ constrain evaluation

$$\llbracket r \rrbracket = \llbracket E_1 \rrbracket \oplus_r \llbracket E_2 \rrbracket \oplus_r \dots$$

*

An algebraic semiring consists of five elements $\langle \mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$, where \mathbb{K} is a domain closed under \oplus and \otimes , \oplus is a binary, associative, commutative operator, \otimes is a binary, associative operator that distributes over \oplus , $\mathbf{0} \in \mathbb{K}$ is the \oplus -identity, and $\mathbf{1} \in \mathbb{K}$ is the \otimes -identity.

Semantics

- Semiring *
- valuation function maps provable ground terms to values
- extended to expressions, e.g. $\llbracket x * y \rrbracket \stackrel{\text{def}}{=} \llbracket x \rrbracket * \llbracket y \rrbracket$
- weighted rules $r \oplus_r E$ constrain evaluation

$$\llbracket r \rrbracket = \llbracket E_1 \rrbracket \oplus_r \llbracket E_2 \rrbracket \oplus_r \dots$$

grounding of a rule body

*

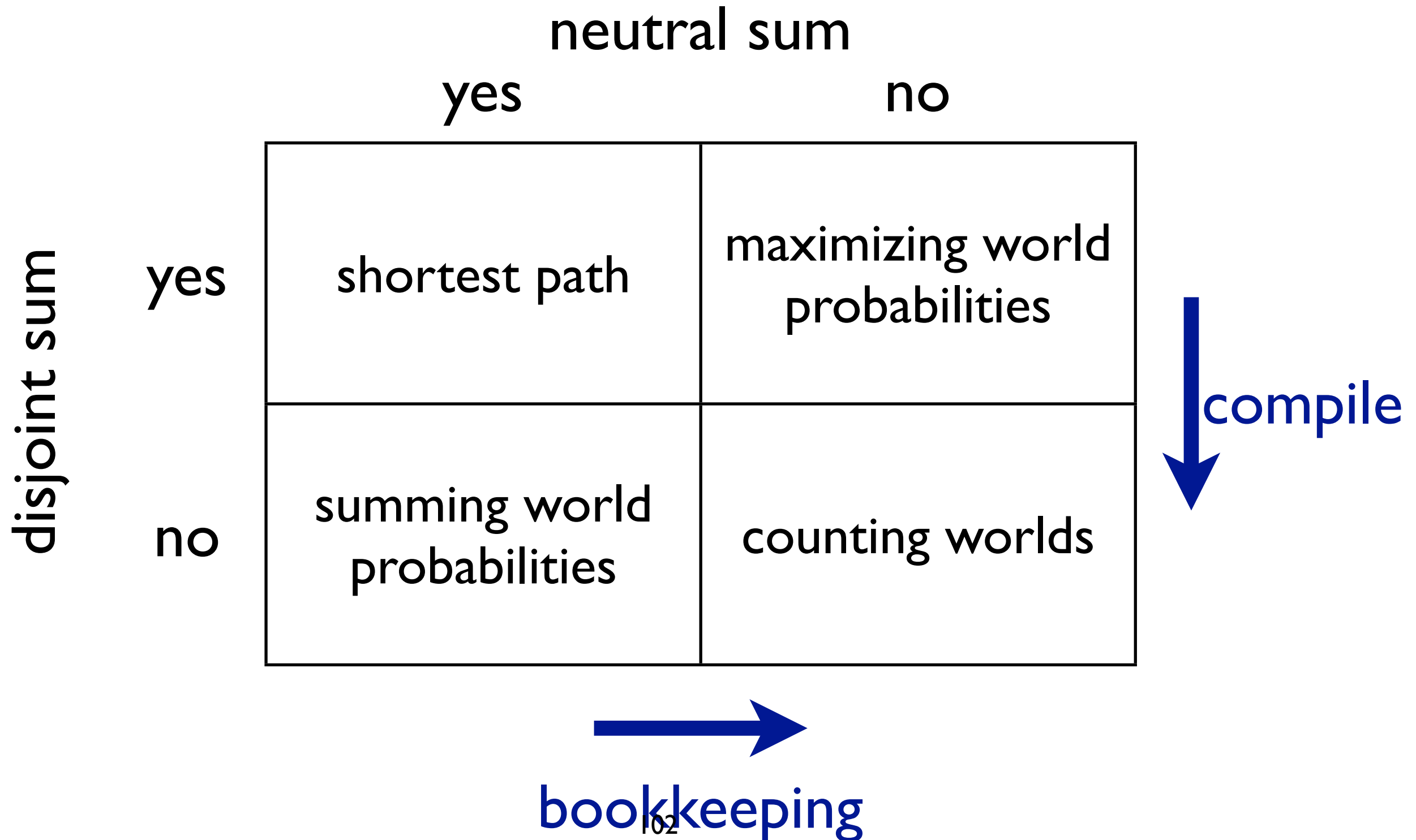
An algebraic semiring consists of five elements $\langle \mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$, where \mathbb{K} is a domain closed under \oplus and \otimes , \oplus is a binary, associative, commutative operator, \otimes is a binary, associative operator that distributes over \oplus , $\mathbf{0} \in \mathbb{K}$ is the \oplus -identity, and $\mathbf{1} \in \mathbb{K}$ is the \otimes -identity.

Algebraic Prolog (aProbLog)

- commutative semiring $(A, \oplus, \otimes, e^{\oplus}, e^{\otimes})$
- algebraic ground literals

$$L(F) = \{f_1, \dots, f_n\} \cup \{\neg f_1, \dots, \neg f_n\}$$
- background knowledge clauses
- labeling function $\alpha: L(F) \rightarrow A$
- $$L(\text{query}) = \bigoplus_{\text{worlds}} \bigotimes_{\text{literals}} \alpha(l)$$

Inference settings



neutral sum

min-sum

yes

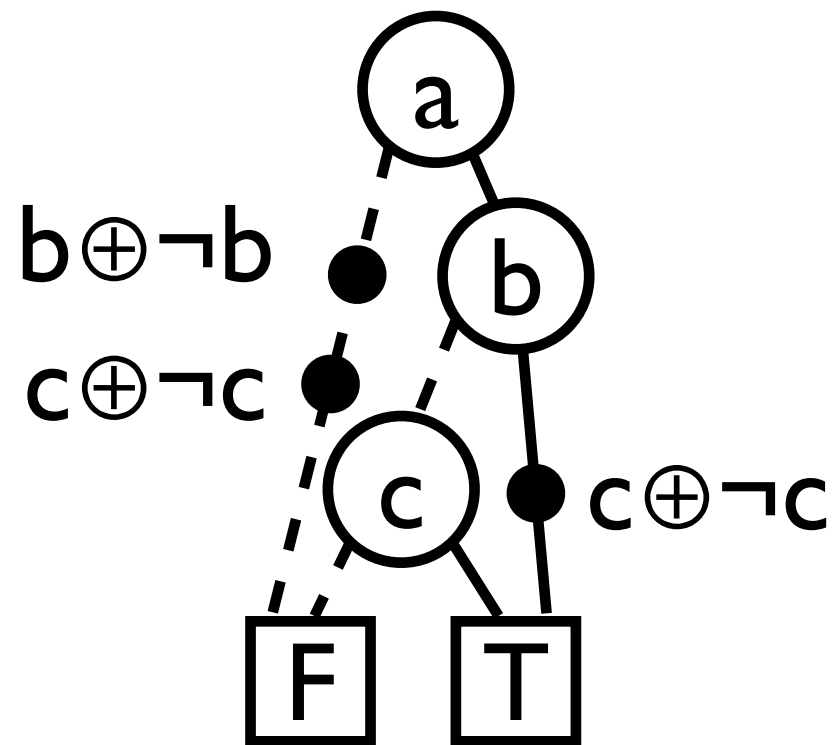
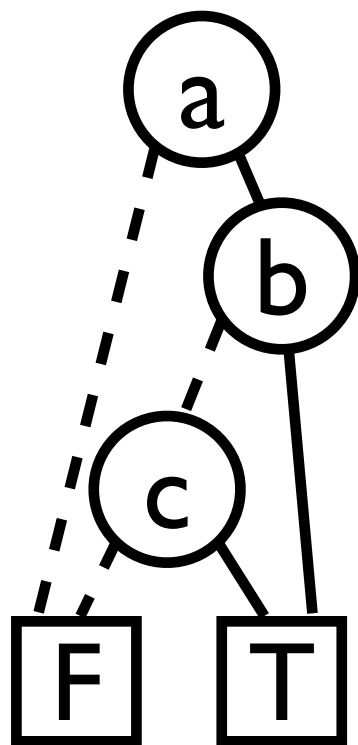
no

most likely

disjoint sum
yes
no

$$(a \wedge b) \\ \vee (a \wedge c)$$

$$(a \wedge b \wedge (c \vee \neg c)) \\ \vee (a \wedge c \wedge (b \vee \neg b))$$



probability

expected cost

filter

expand

Query Evaluation in PDB

Query Evaluation in PDB

- **Extensional** evaluation
 - guided by query expression only
 - exploit DB technology
 - for queries known to have polytime evaluation

Query Evaluation in PDB

- **Extensional** evaluation
 - guided by query expression only
 - exploit DB technology
 - for queries known to have polytime evaluation
- **Intensional** evaluation
 - construct **lineage** (= propositional formula)
 - compute probability of lineage
 - all queries

knowledge
compilation

Approximate Inference

- Lower and upper bounds

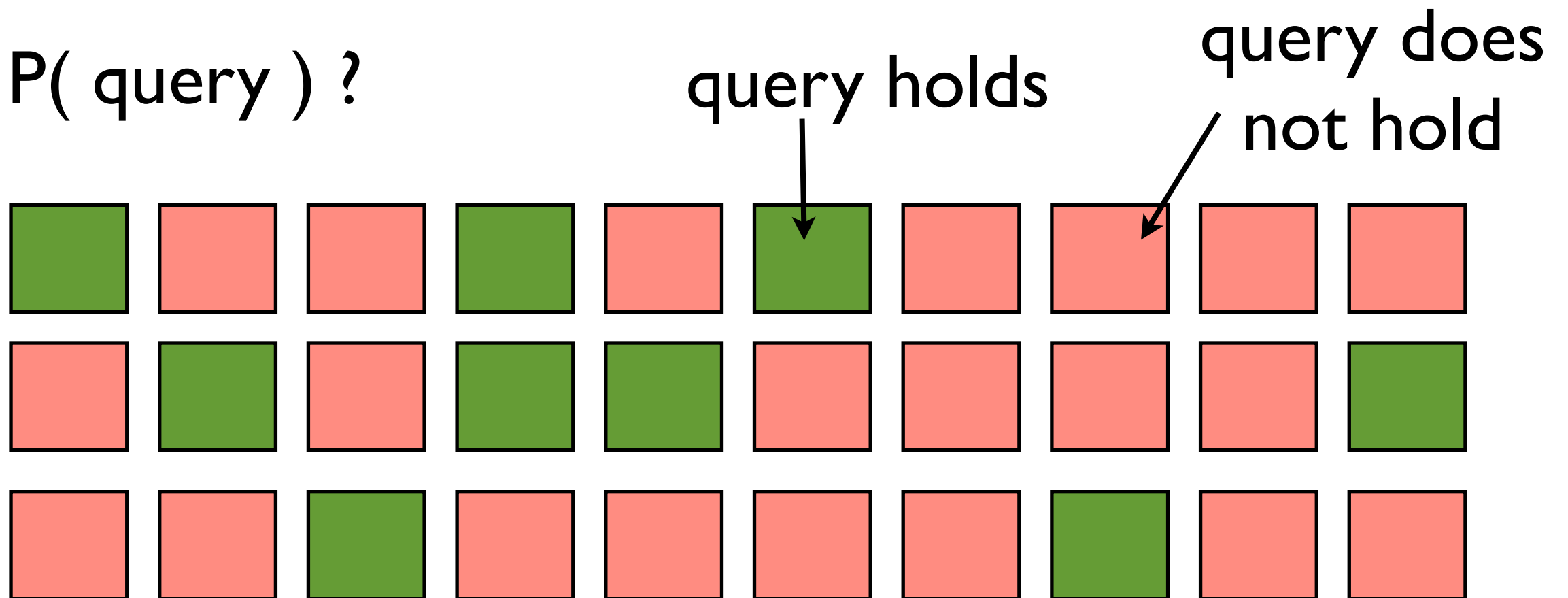
$$\phi_L \models \phi \models \phi_U$$

$$P(\phi_L) \leq P(\phi) \leq P(\phi_U)$$

- Sampling

Sampling

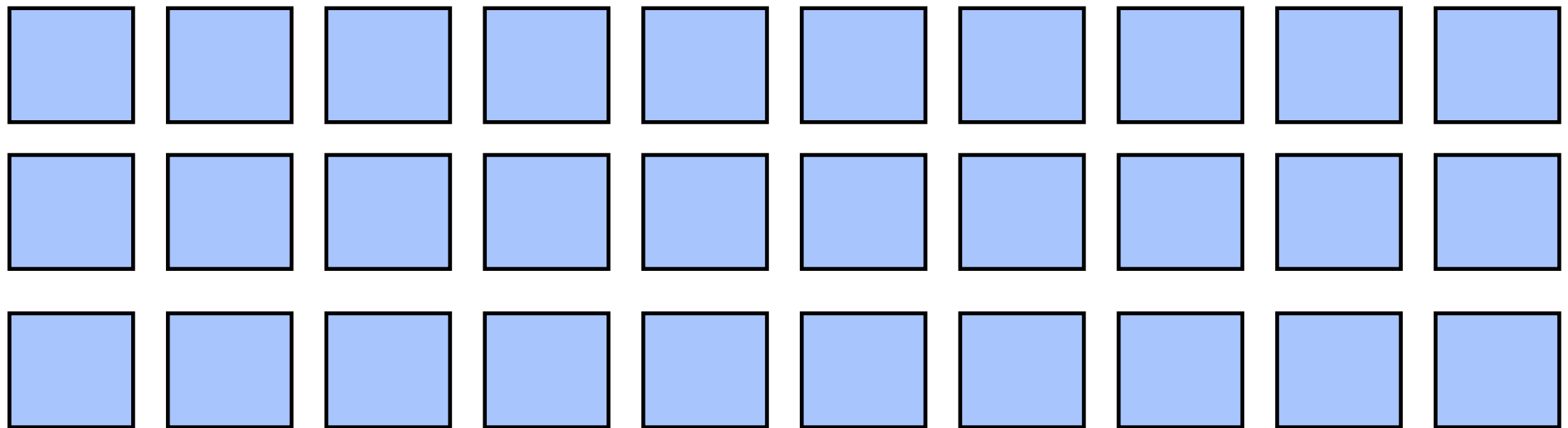
- $P(\text{query})$?



$$P(\text{query}) \approx \frac{\# \text{ query holds}}{\# \text{ worlds sampled}}$$

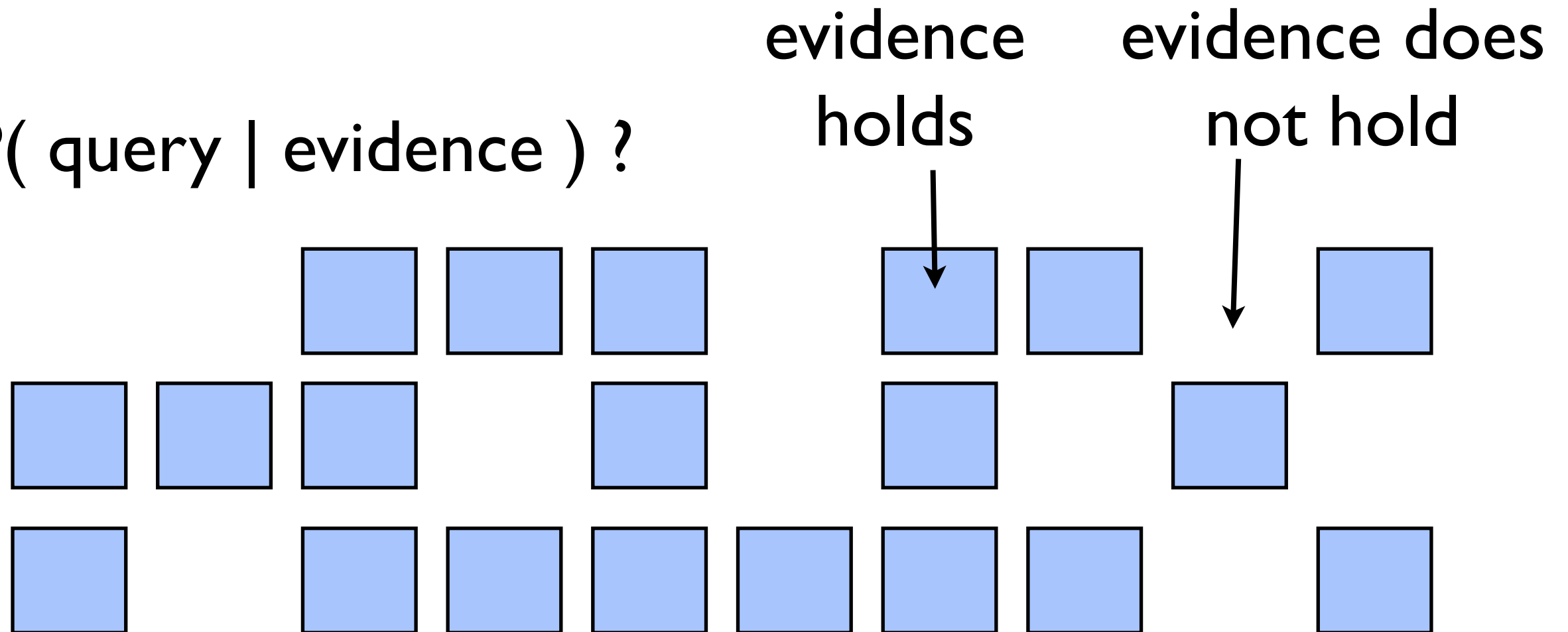
Rejection Sampling

- $P(\text{query} \mid \text{evidence})$?



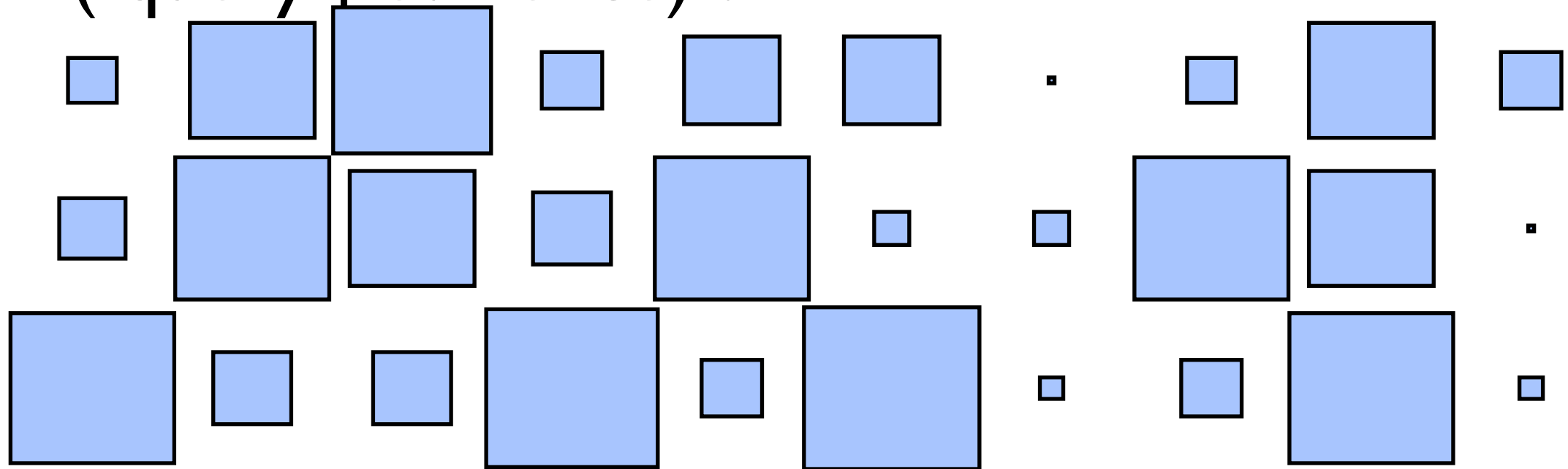
Rejection Sampling

- $P(\text{query} \mid \text{evidence})$?



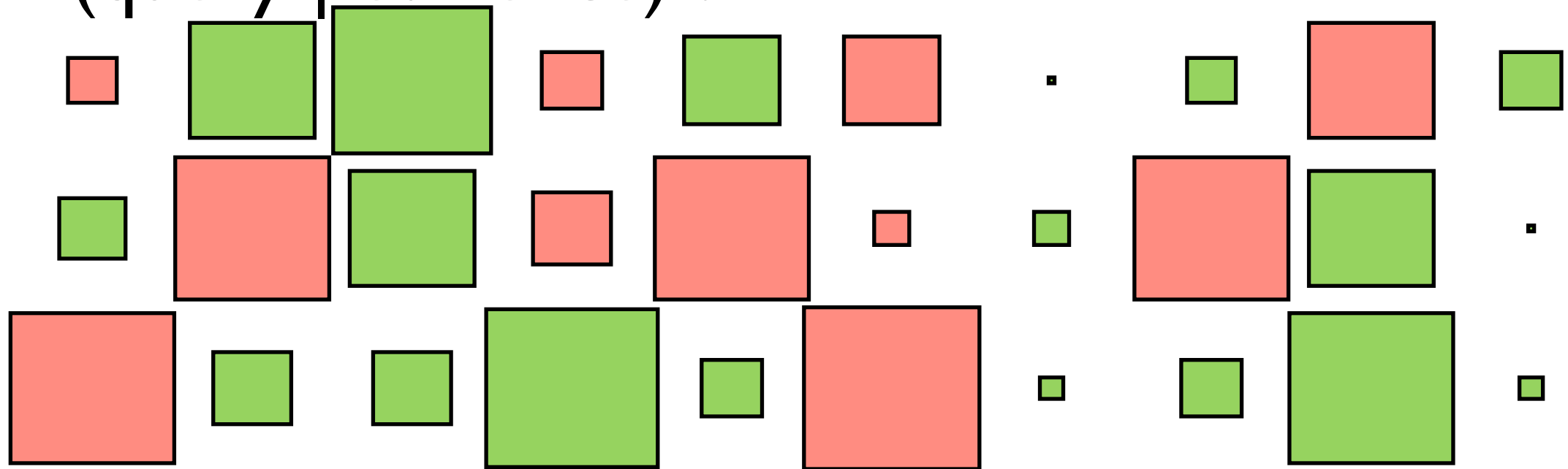
Likelihood Weighting

- $P(\text{query} \mid \text{evidence})$?



Likelihood Weighting

- $P(\text{query} \mid \text{evidence})$?



Markov Chain Monte Carlo (MCMC)

- Generate next sample by modifying current one
- Most common inference approach for PP languages such as Church, BLOG, ...
- Also considered for PRISM and ProbLog

Key challenges:

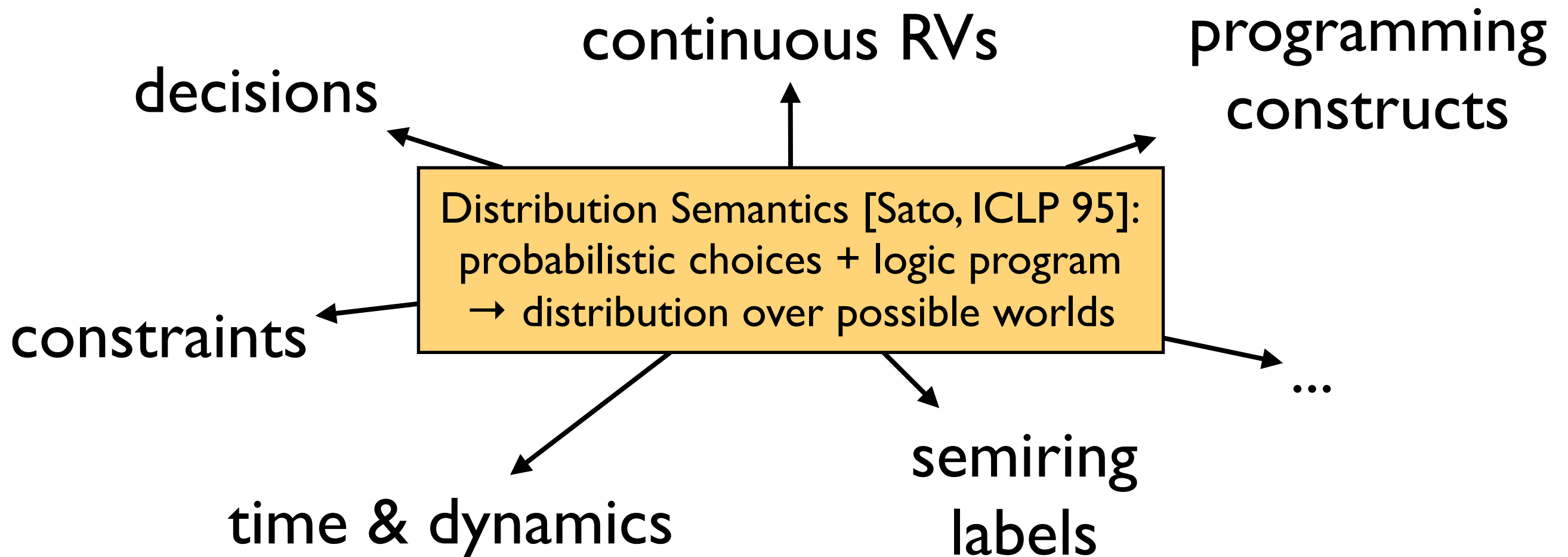
- how to propose next sample
- how to handle evidence

Roadmap

- Modeling
- Reasoning
- Language extensions
- Advanced topics

... with some detours on the way

Extensions of basic PLP



Dynamics

Dynamics: Evolving Networks



- *Travian*: A massively multiplayer real-time strategy game
 - Commercial game run by TravianGames GmbH
 - ~3.000.000 players spread over different “worlds”
 - ~25.000 players in one world

[Thon et al. ECML 08, MLJ 11]



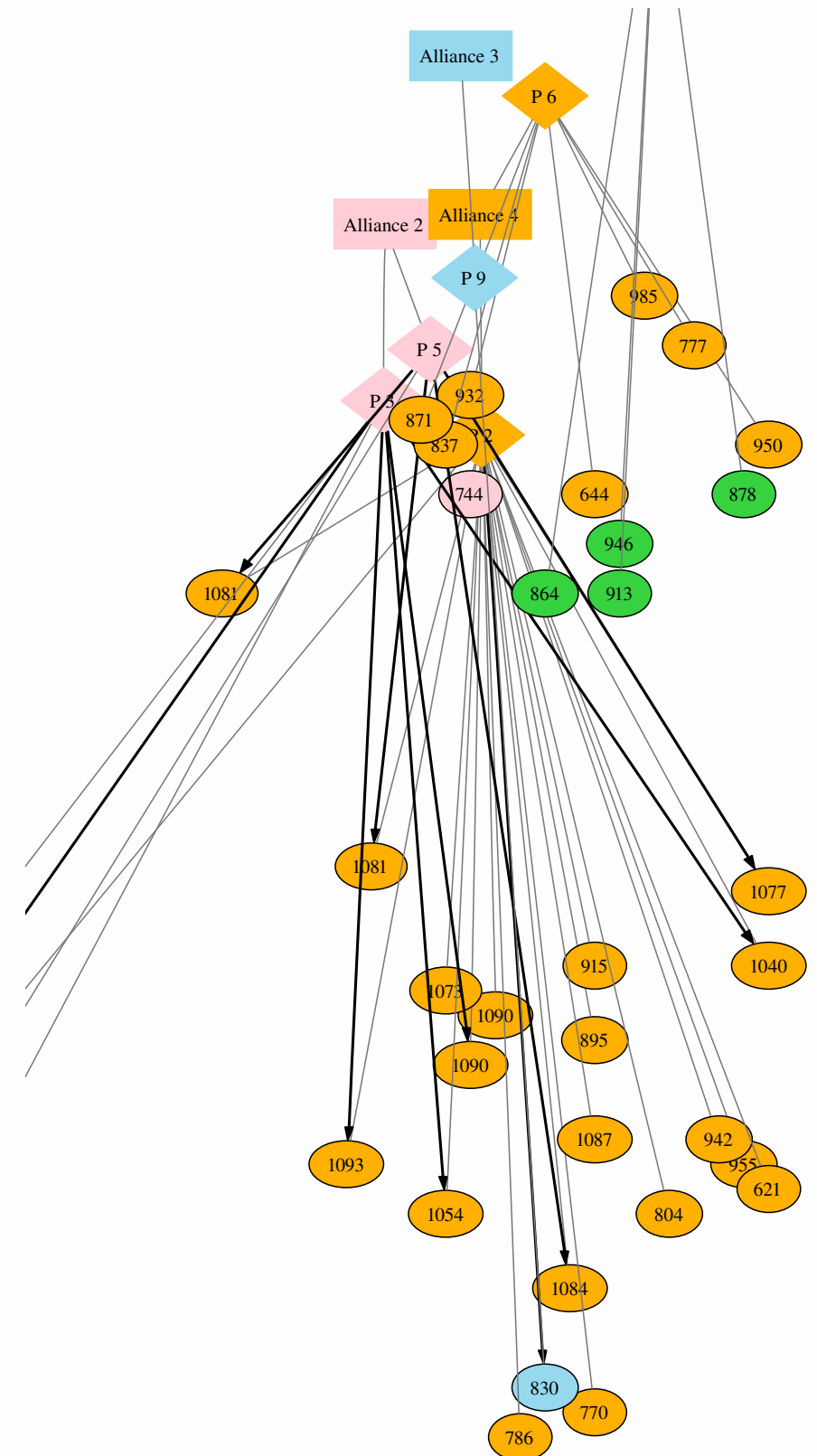
World Dynamics

Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?



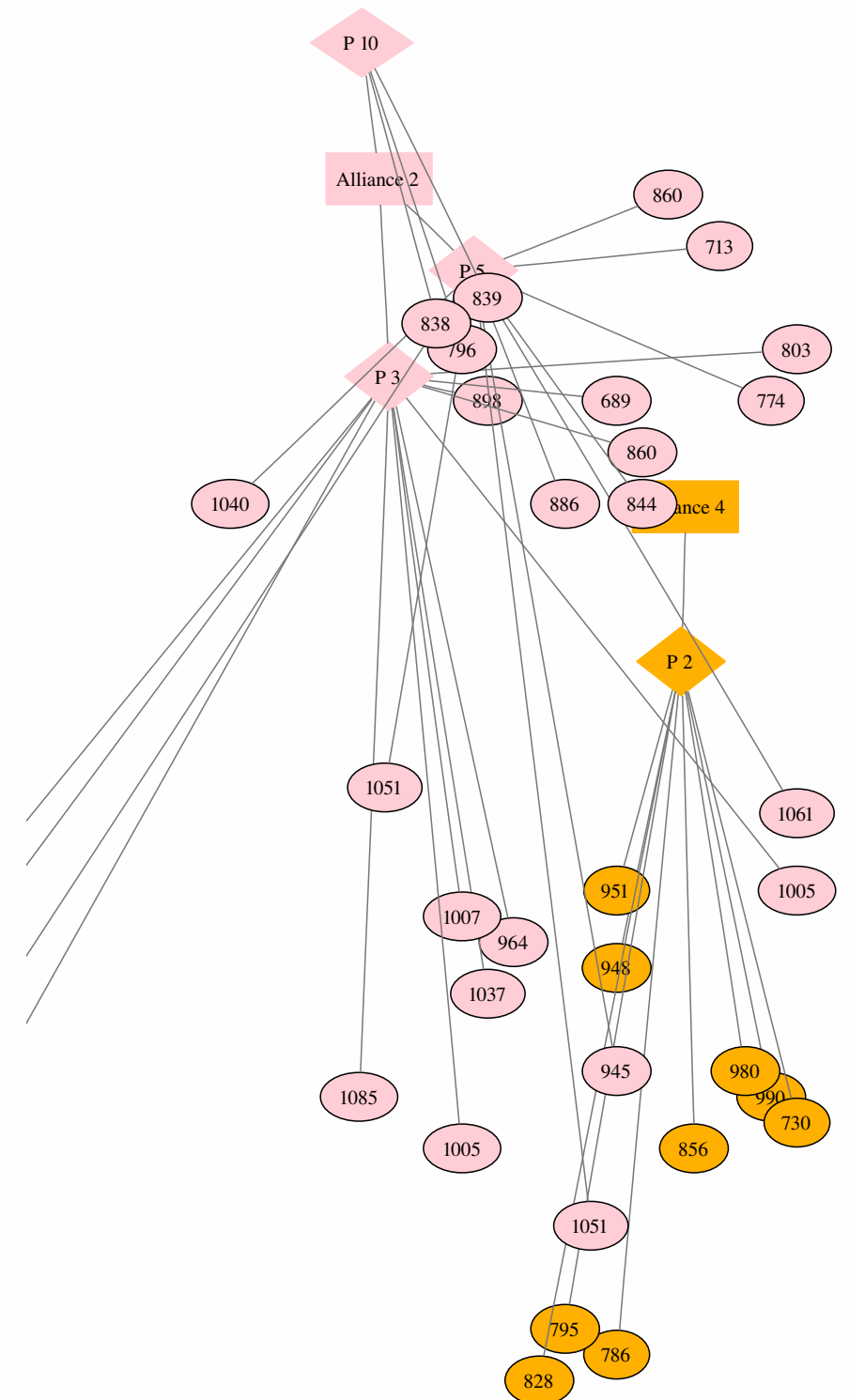
World Dynamics

Fragment of world with

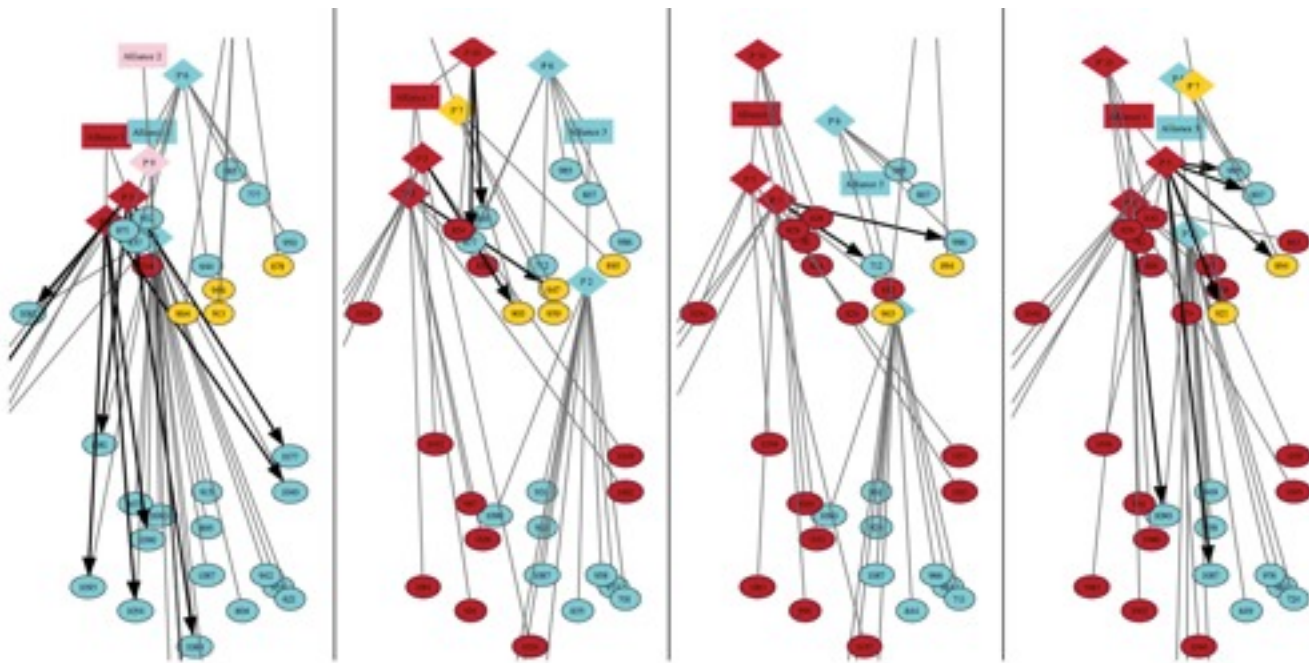
~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

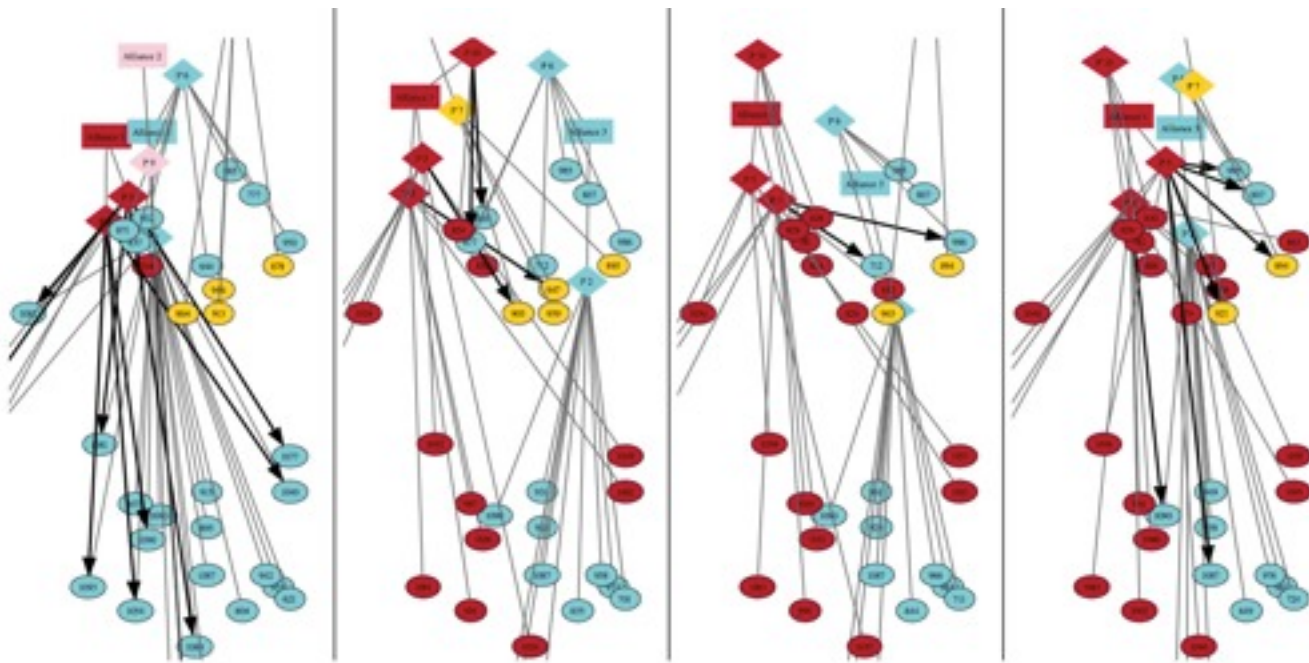


Causal Probabilistic Time-Logic (CPT-L)



how does the
world change
over time?

Causal Probabilistic Time-Logic (CPT-L)



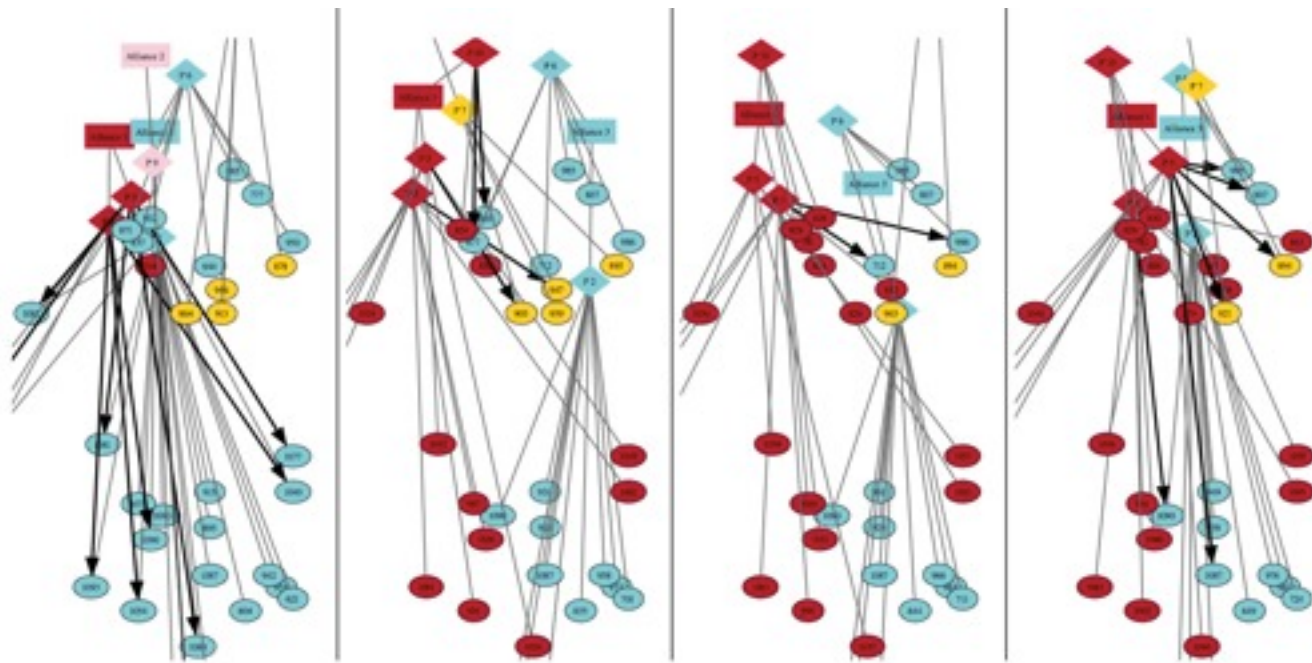
how does the
world change
over time?

```
0.4 :: conquest (Attacker, C) ; 0.6 :: nil <-
```

```
city (C, Owner) , city (C2, Attacker) , close (C, C2) .
```

if **cause** holds at time T

Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

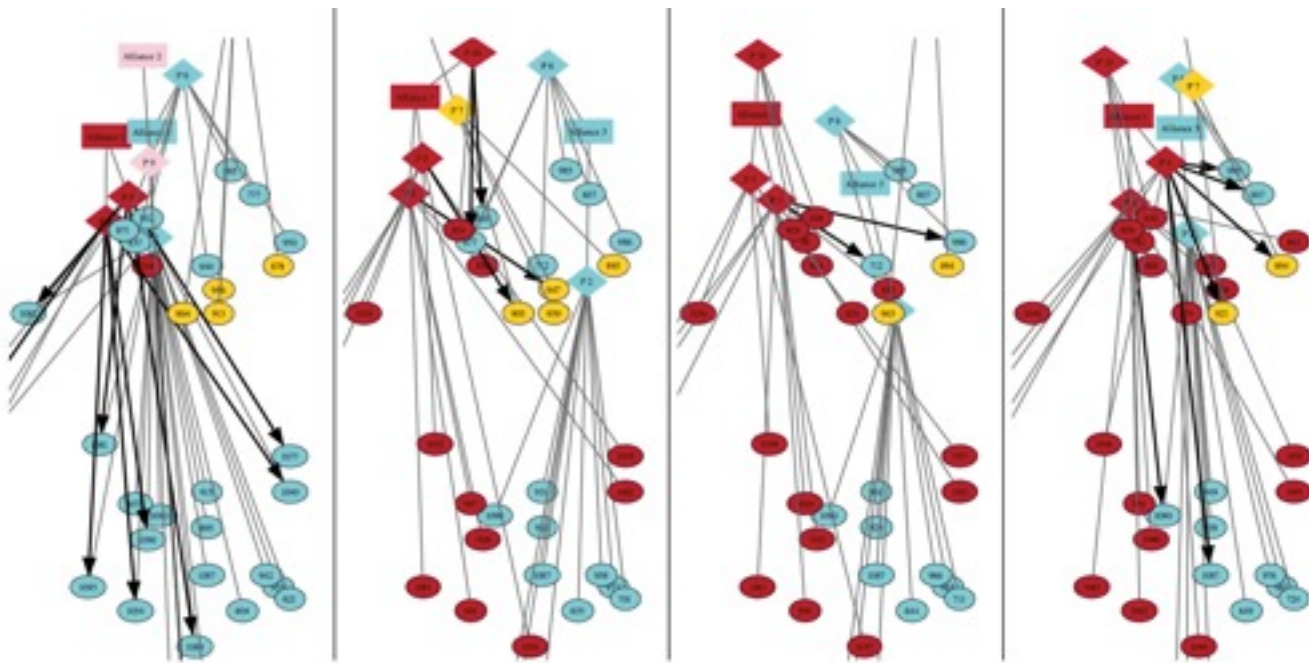
one of the **effects** holds at time $T+1$

```
0.4 :: conquest (Attacker, C) ; 0.6 :: nil <-
```

```
city (C, Owner) , city (C2, Attacker) , close (C, C2) .
```

if **cause** holds at time T

Causal Probabilistic Time-Logic (CPT-L)



how does the
world change
over time?

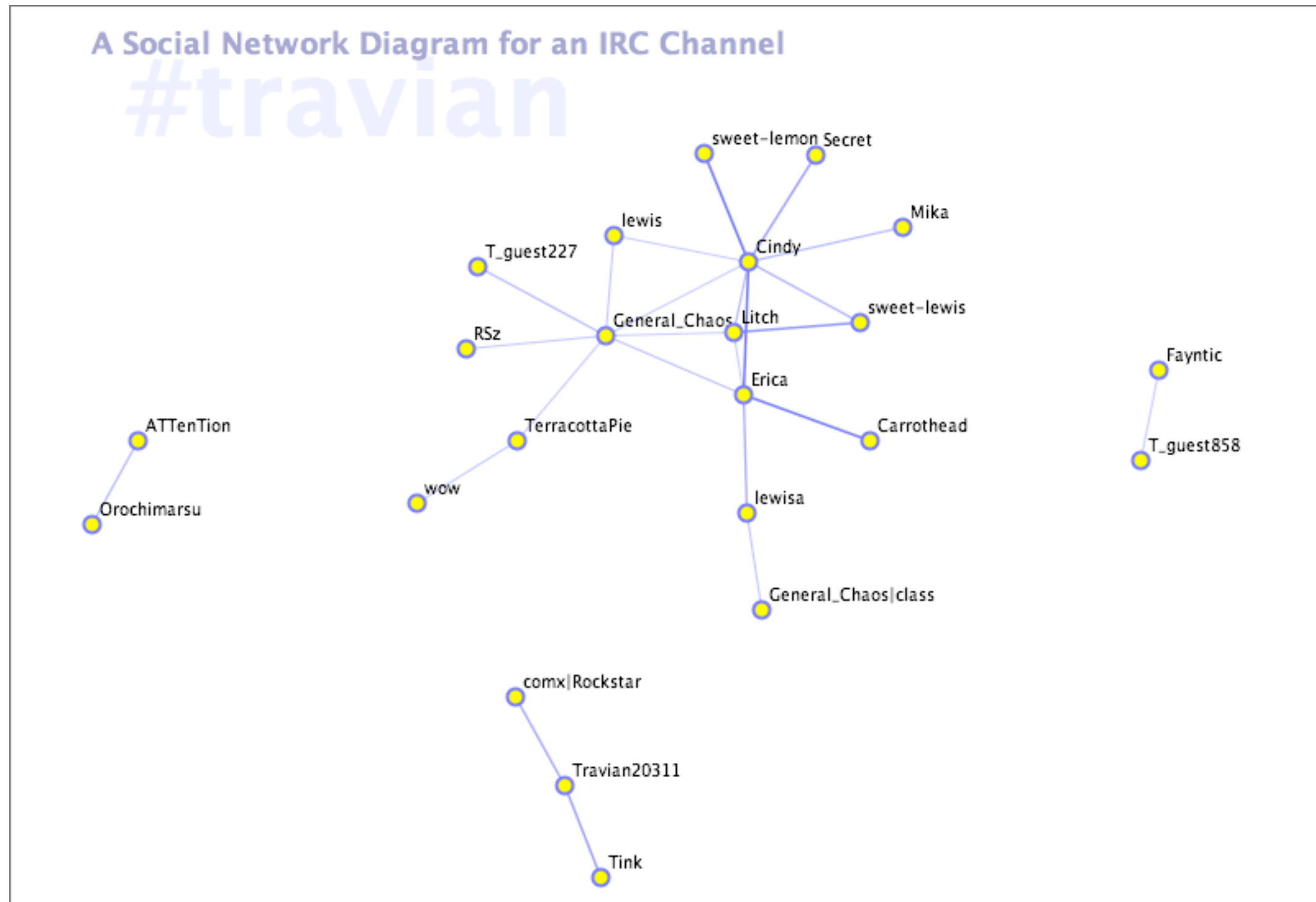
one of the **effects** holds at time $T+1$

```
0.4 :: conquest (Attacker, C) ; 0.6 :: nil <-
```

```
city (C, Owner) , city (C2, Attacker) , close (C, C2) .
```

if **cause** holds at time T

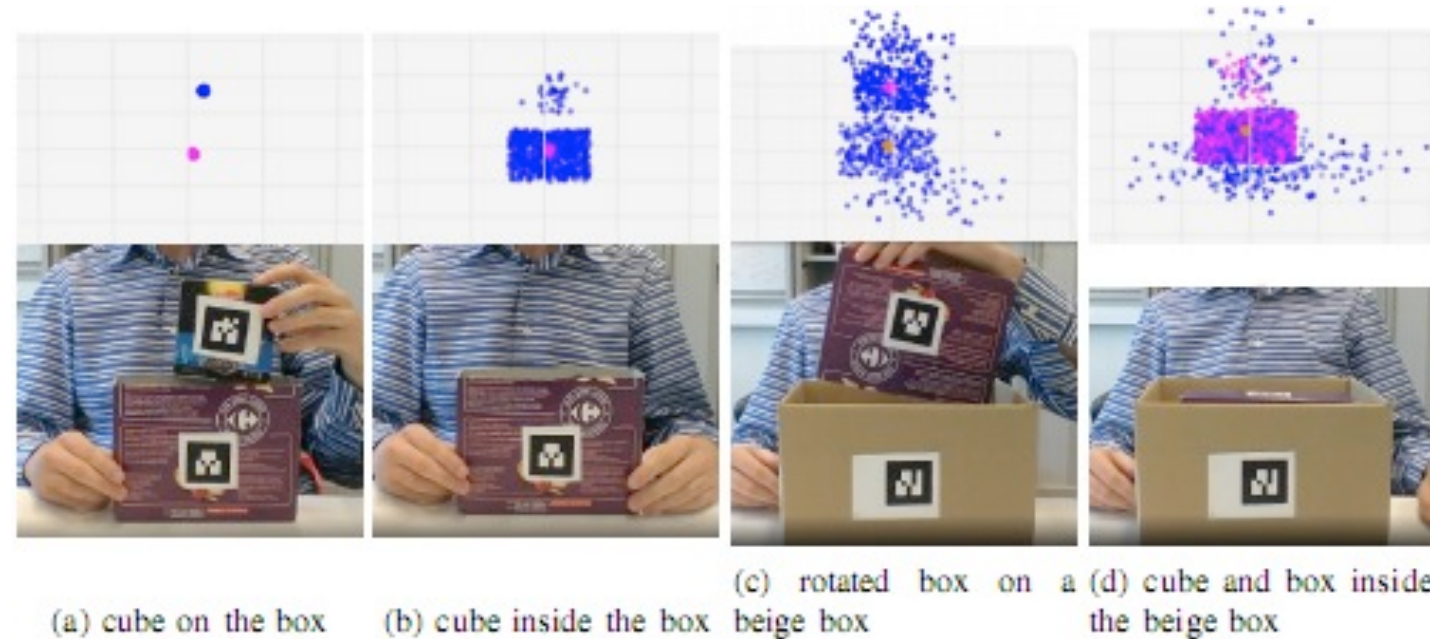
Social Network of Chats



Relational State Estimation over Time

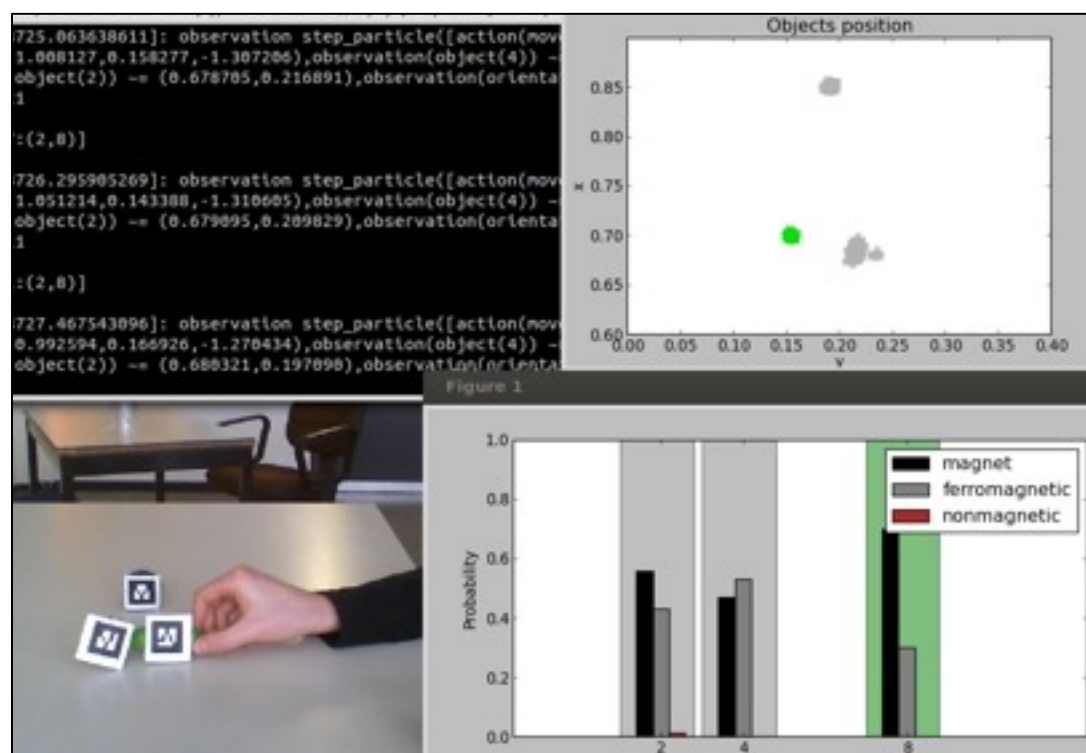
Magnetism scenario

- object tracking
- category estimation from interactions



Box scenario

- object tracking even when invisible
- estimate spatial relations

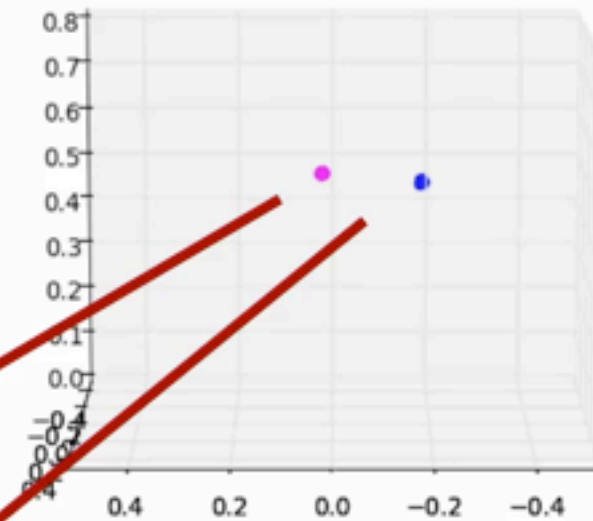


Speed 0x

Queries (updated every 5 steps)

```
[ ]  
on(X,Y):  
[1.0:(3,(table)),1.0:(4,(table))]  
inside(X,Y):  
[ ]  
tr_inside(X,Y):  
[ ]
```

Particles



Box ID=4

Cube ID=3

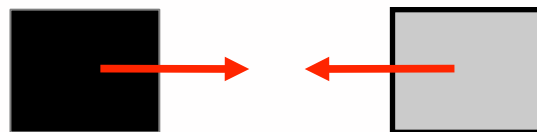
IROS 13

Magnetic scenario

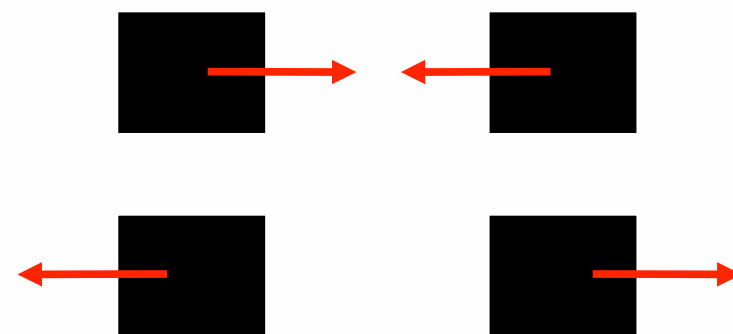
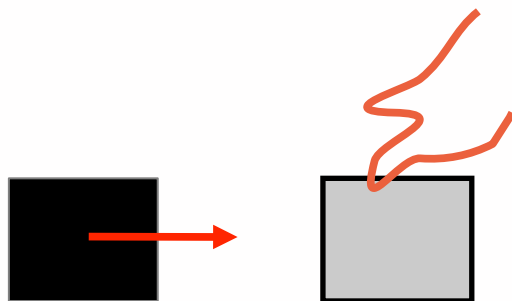
- 3 object types: magnetic, ferromagnetic, nonmagnetic

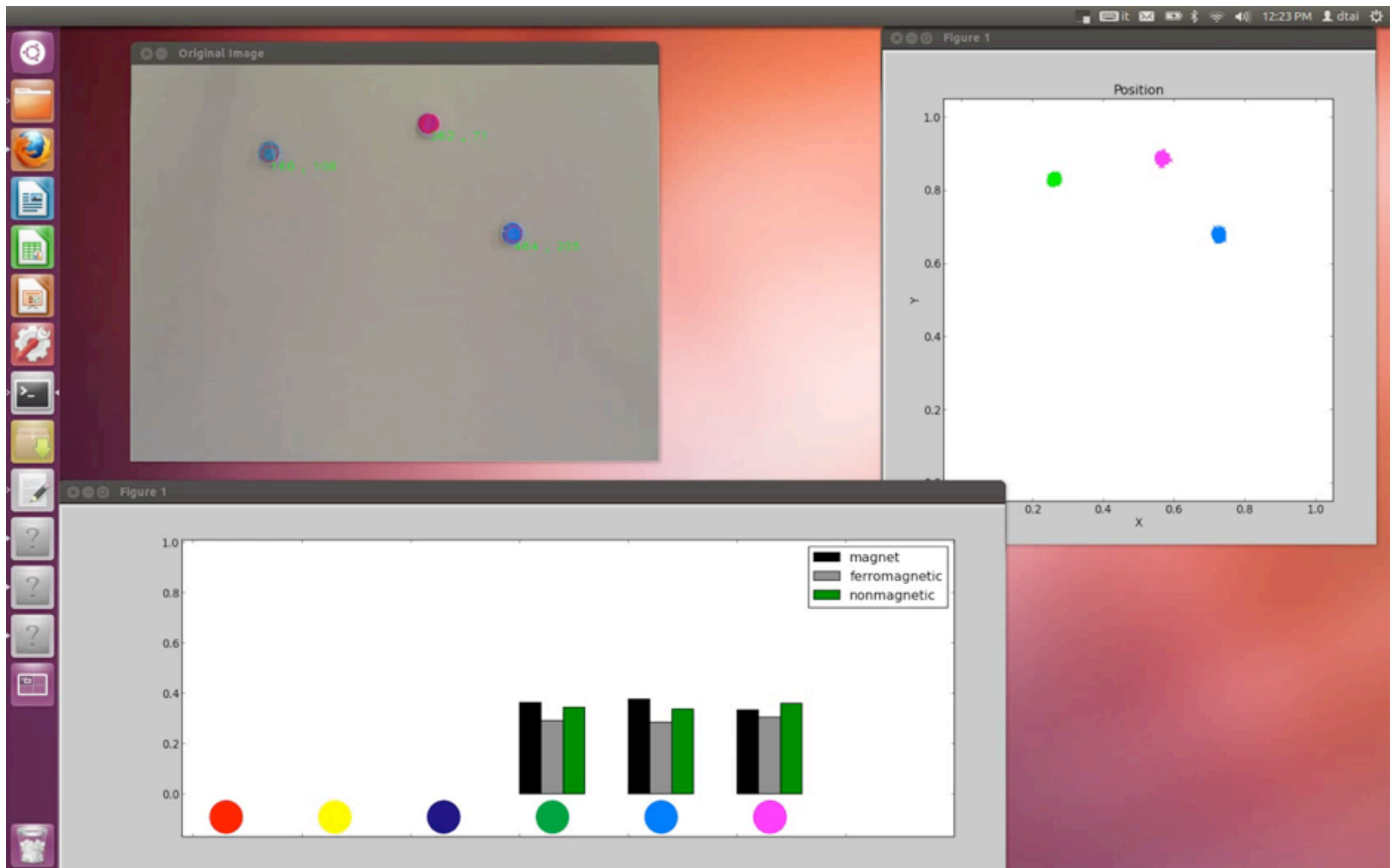


- Nonmagnetic objects do not interact
- A magnet and a ferromagnetic object attract each other

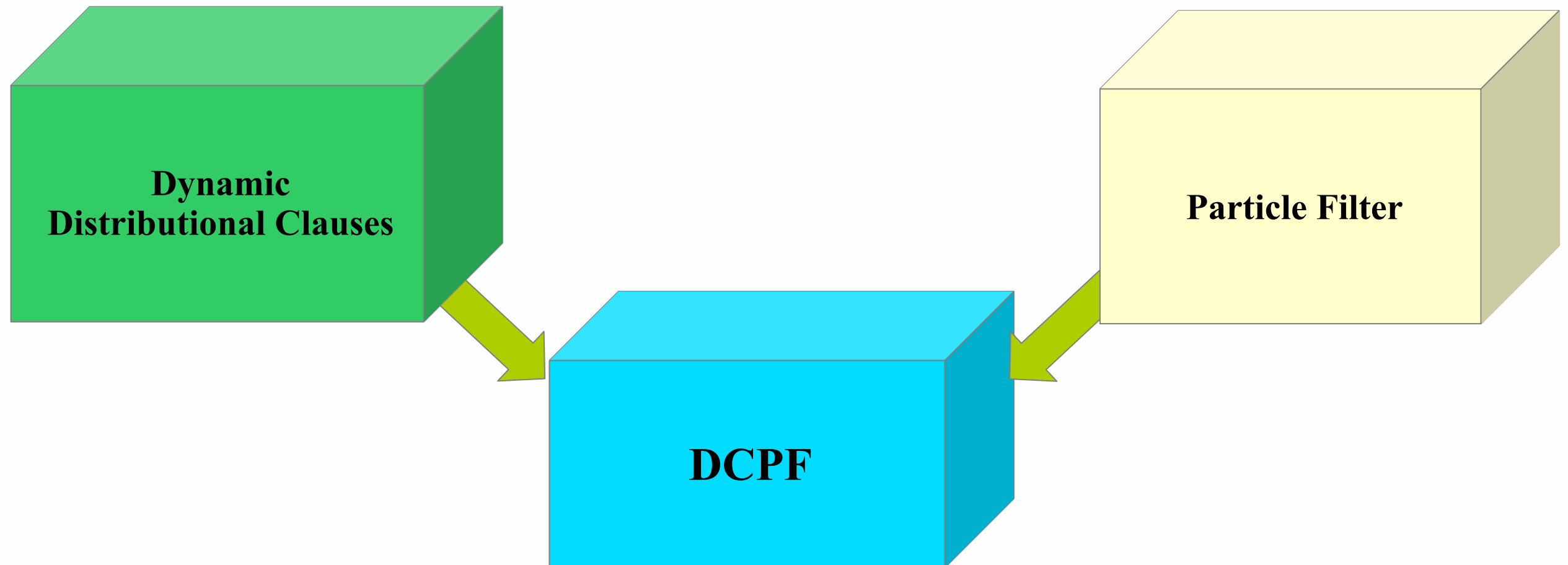


- Magnetic force that depends on the distance
- If an object is held magnetic force is compensated.





DC Particle Filter (DCPF)



Goal {
Flexible (relational) state representation
Fast inference (state estimation) in general models

“A particle filter for hybrid relational domains” IROS 2013
D. Nitti, T. De Laet, L. De Raedt

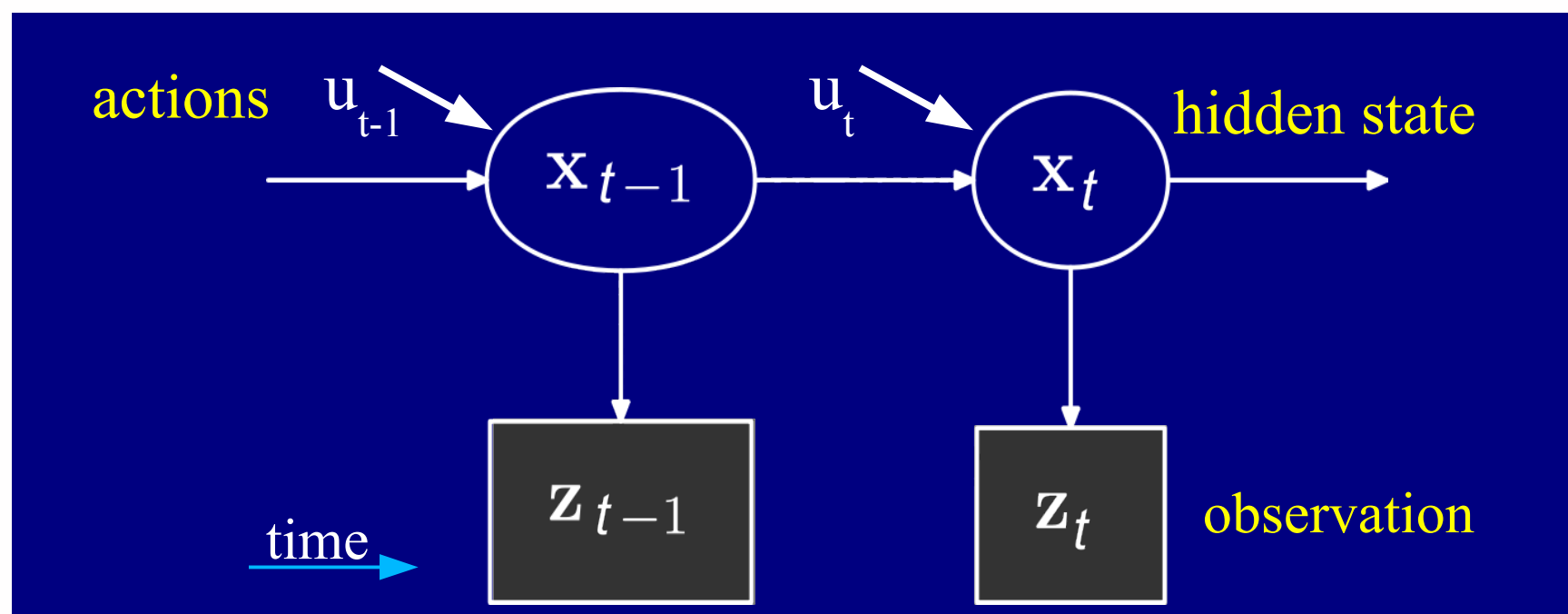
Dynamic Distributional Clauses

Prior distribution $p(x_0)$

State transition model $p(x_t|x_{t-1},u_t)$

Measurement model $p(z_t|x_t)$

Other rules: $p(x'_t|x''_t)$



Magnetic scenario

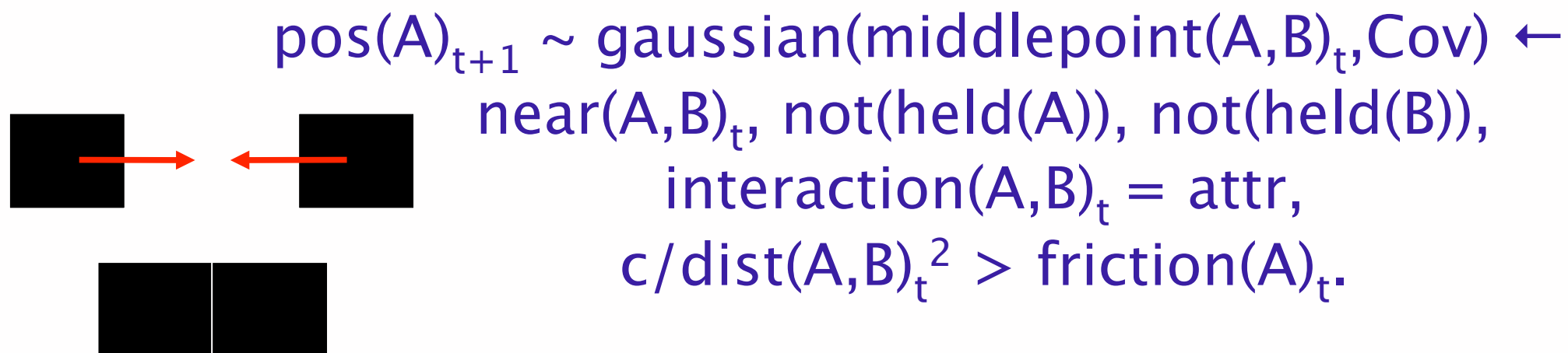
- 3 object types: magnetic, ferromagnetic, nonmagnetic

$\text{type}(X)_t \sim \text{finite}([1/3:\text{magnet}, 1/3:\text{ferromagnetic}, 1/3:\text{nonmagnetic}]) \leftarrow \text{object}(X).$

- 2 magnets attract or repulse

$\text{interaction}(A,B)_t \sim \text{finite}([0.5:\text{attraction}, 0.5:\text{repulsion}]) \leftarrow \text{object}(A), \text{object}(B), A < B, \text{type}(A)_t = \text{magnet}, \text{type}(B)_t = \text{magnet}.$

- Next position after attraction

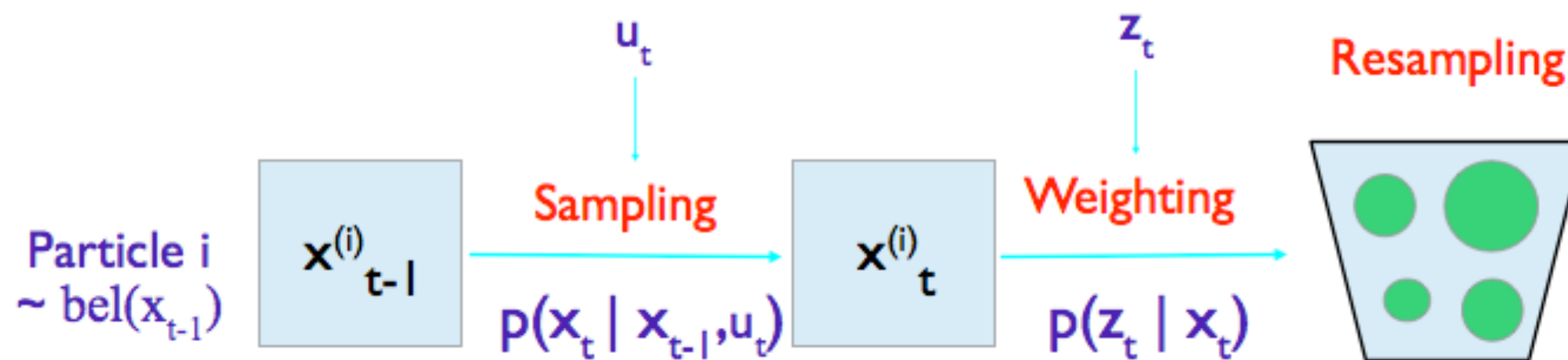


$\text{pos}(A)_{t+1} \sim \text{gaussian}(\text{pos}(A)_t, \text{Cov}) \leftarrow \text{not}(\text{attraction}(A,B)).$

Particle Filter

(Sequential Monte Carlo)

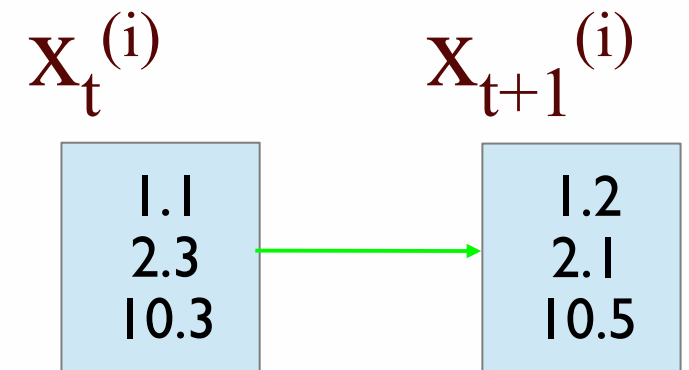
- Based on sampling \rightarrow approximate inference
- Particles (samples) to represent $\text{bel}(x_t)$



Classical Particle Filter vs DCPF

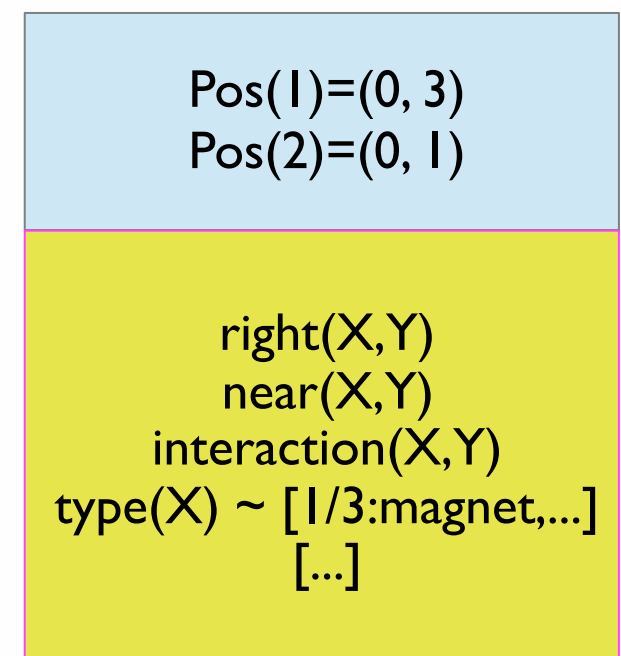
- Classical PF

- Fixed set of random variables
- Update the entire state



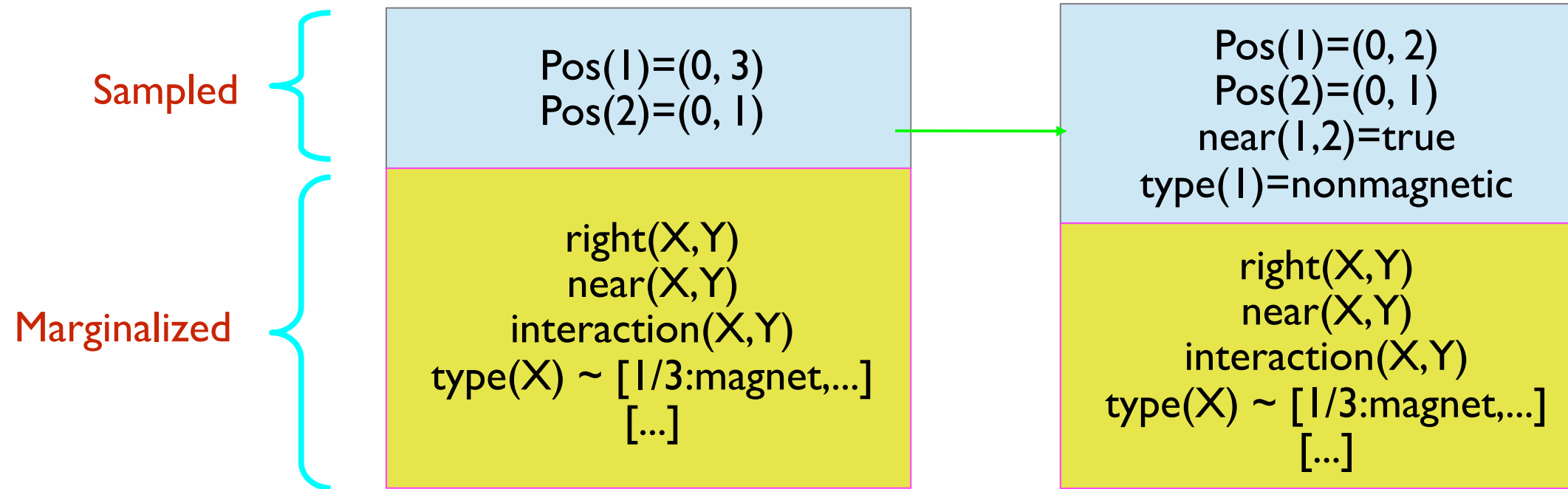
- DCPF

- **Adaptive state** (particle): the number of facts / random variables can change over time
- Particles are partial interpretations
- Expressive language

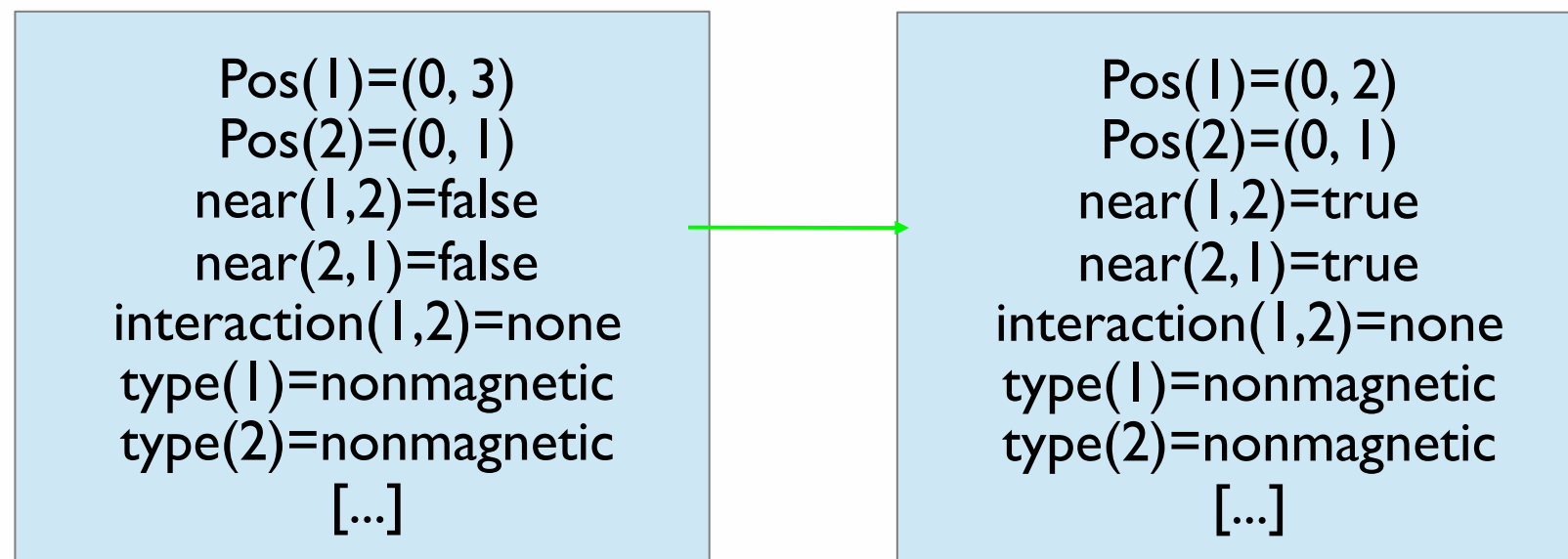


Optimized inference: partial state

Distributional Clauses Particle Filter (DCPF)

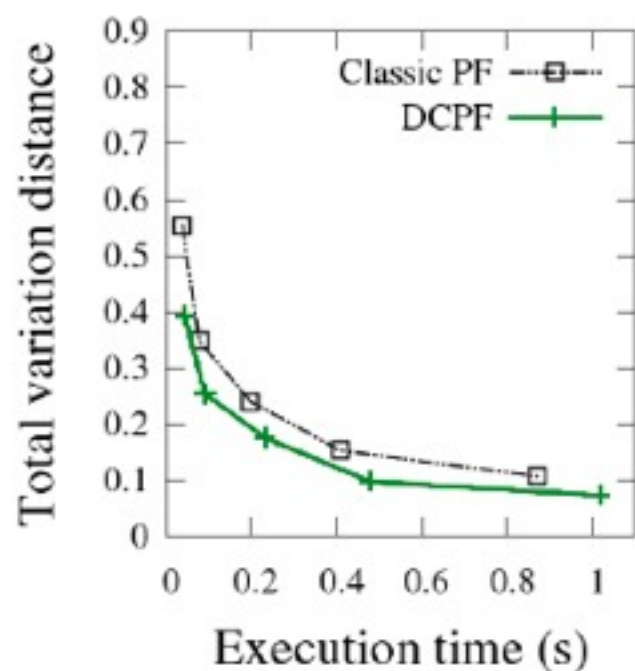


Classical particle filter

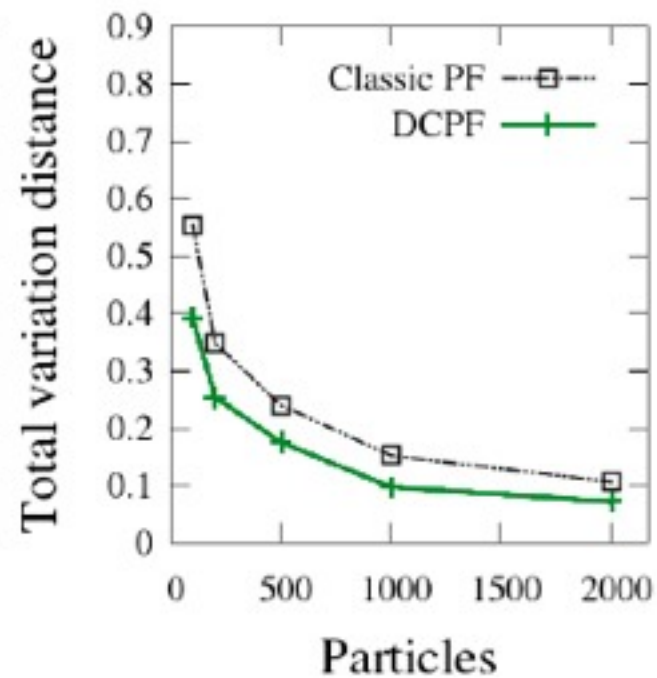


Experiments

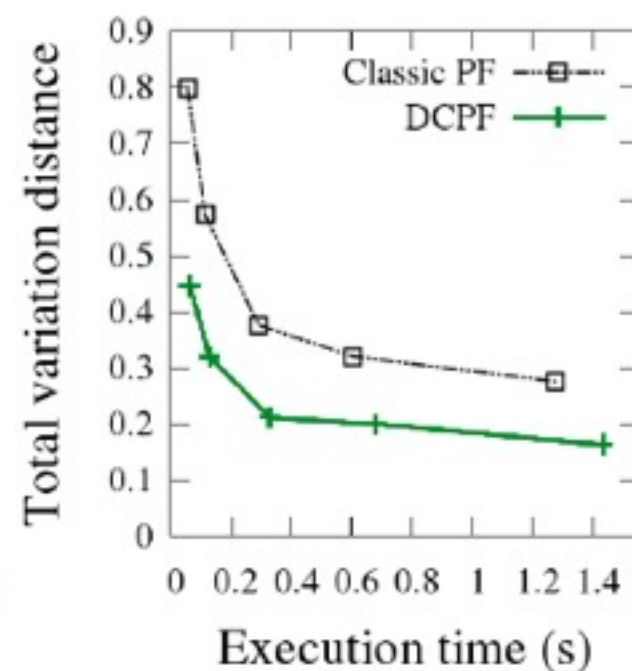
- Particles are partial state, remaining variables are marginalized
- Better performance in bigger models



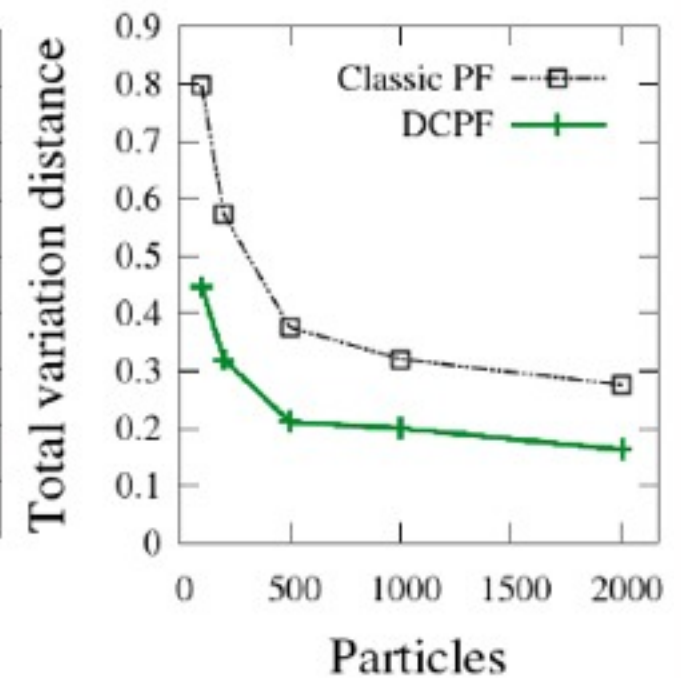
(a) 3 objects



(b) 3 objects



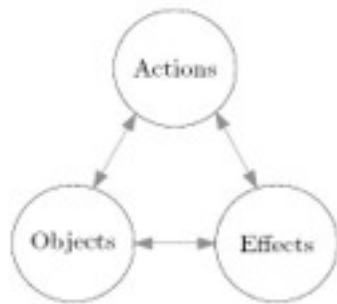
(c) 4 objects



(d) 4 objects

Learning relational affordances

Learn probabilistic model

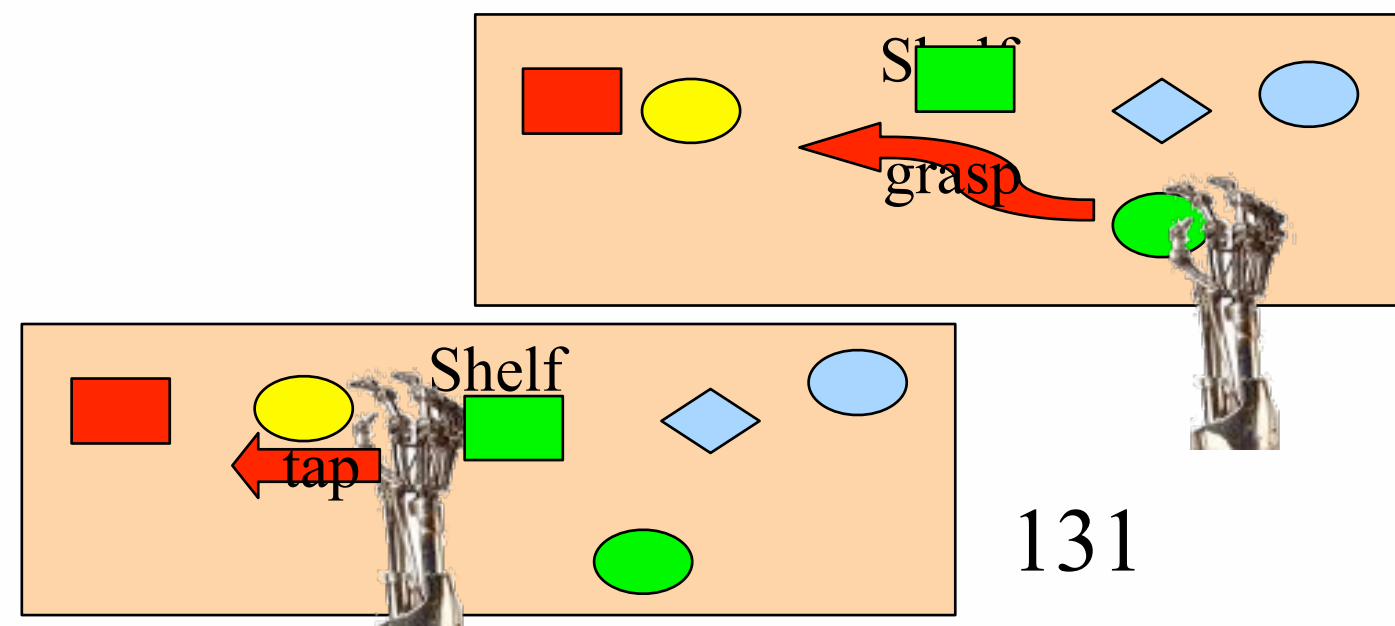
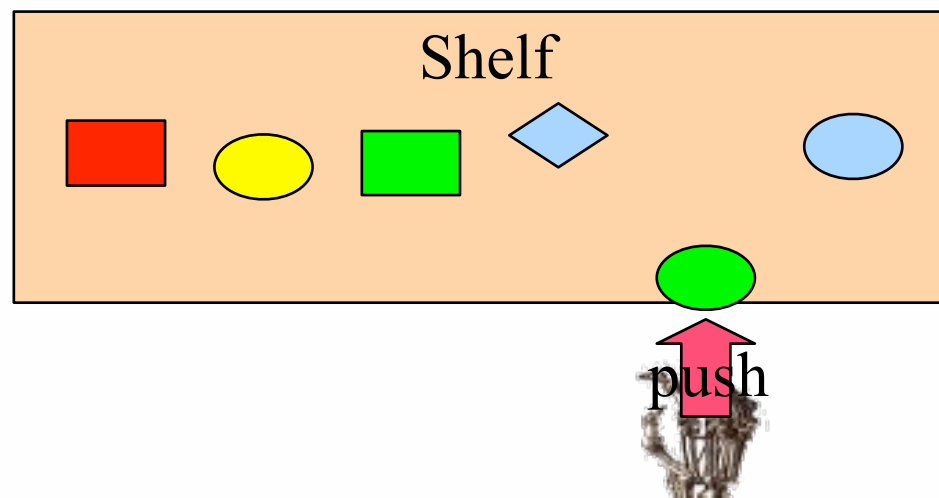


Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

Learning relational
affordances
between
two objects
(learnt by experience)

From two object interactions
Generalize to N

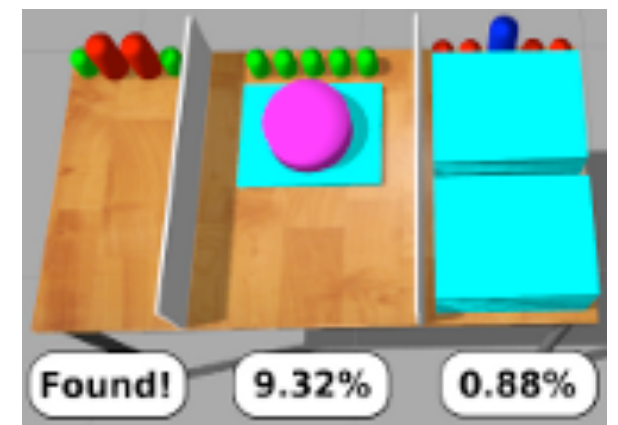
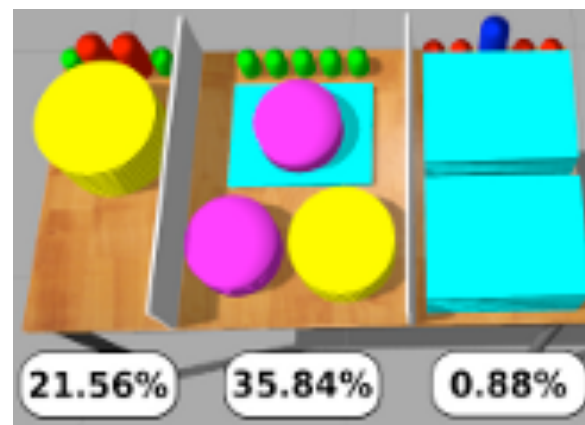
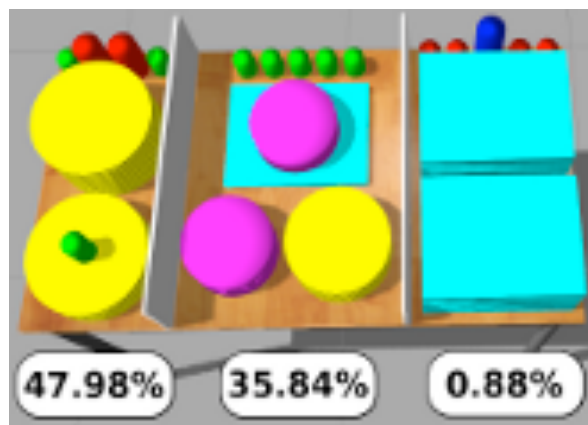
Moldovan et al. ICRA 12, 13, 14



Occluded Object Search



- DC model of objects and their spatial arrangement
- different types of objects suitable for different tasks
- shelves with objects of different shape and size
- given a task, find an object to perform that task



ProbLog for activity recognition from video



CAVIAR-INRIA human
activity dataset

28 videos
 ≈ 26.500 frames

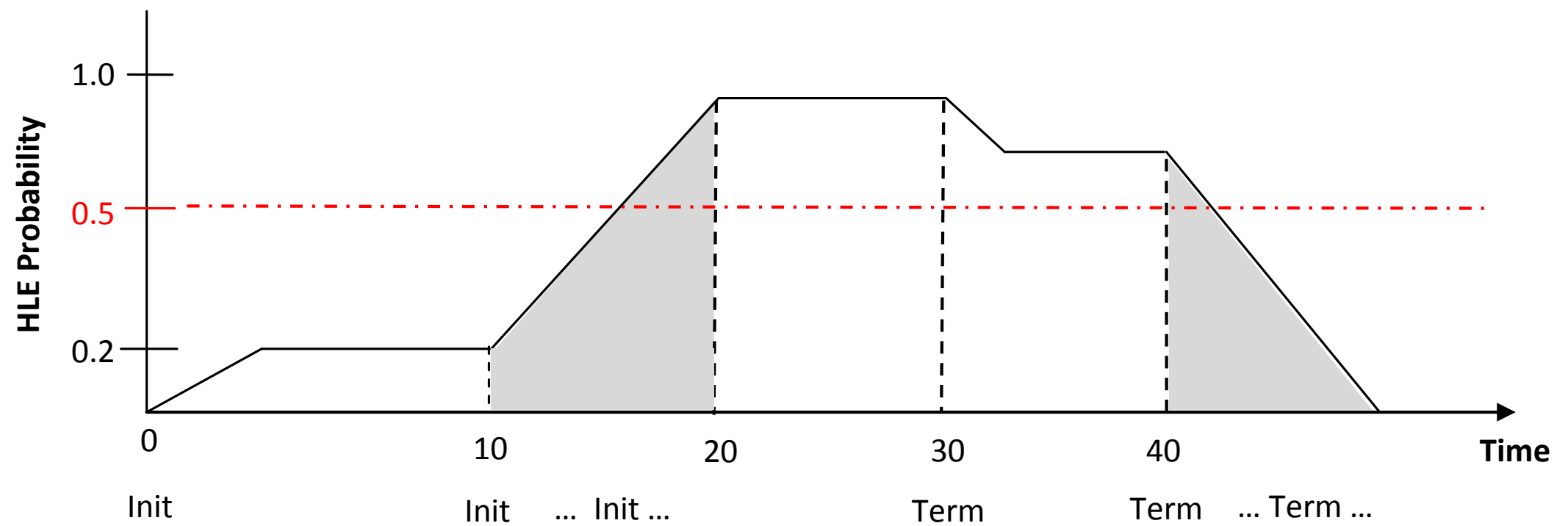
- Separation between low-level events (LLE) and high-level events (HLE)
 - LLE: *walking, running, active, inactive, abrupt*
 - HLE: *meeting, moving, fighting, leaving_object*
- Probabilistic Logic approach: *Event Calculus in ProbLog* (Prob-EC) to infer the high-level events from an **algebra** of low-level events.
- Example:

$$\begin{aligned} \text{initiatedAt}(\text{fighting}(P_1, P_2) = \text{true}, T) \leftarrow \\ \text{happensAt}(\text{abrupt}(P_1), T), \\ \text{holdsAt}(\text{close}(P_1, P_2, 44) = \text{true}, T), \\ \text{not happensAt}(\text{inactive}(P_2), T). \end{aligned}$$

Event Recognition using ProbLog

Input	Output
340 0.45 :: <i>inactive</i> (id_0)	340 0.41 :: <i>leaving_object</i> (id_1, id_0)
340 0.80 :: $p(id_0) = (20.88, -11.90)$	340 0.55 :: <i>moving</i> (id_2, id_3)
340 0.55 :: <i>appear</i> (id_0)	
340 0.15 :: <i>walking</i> (id_2)	
340 0.80 :: $p(id_2) = (25.88, -19.80)$	
340 0.25 :: <i>active</i> (id_1)	
340 0.66 :: $p(id_1) = (20.88, -11.90)$	
340 0.70 :: <i>walking</i> (id_3)	
340 0.46 :: $p(id_3) = (24.78, -18.77)$	
...	

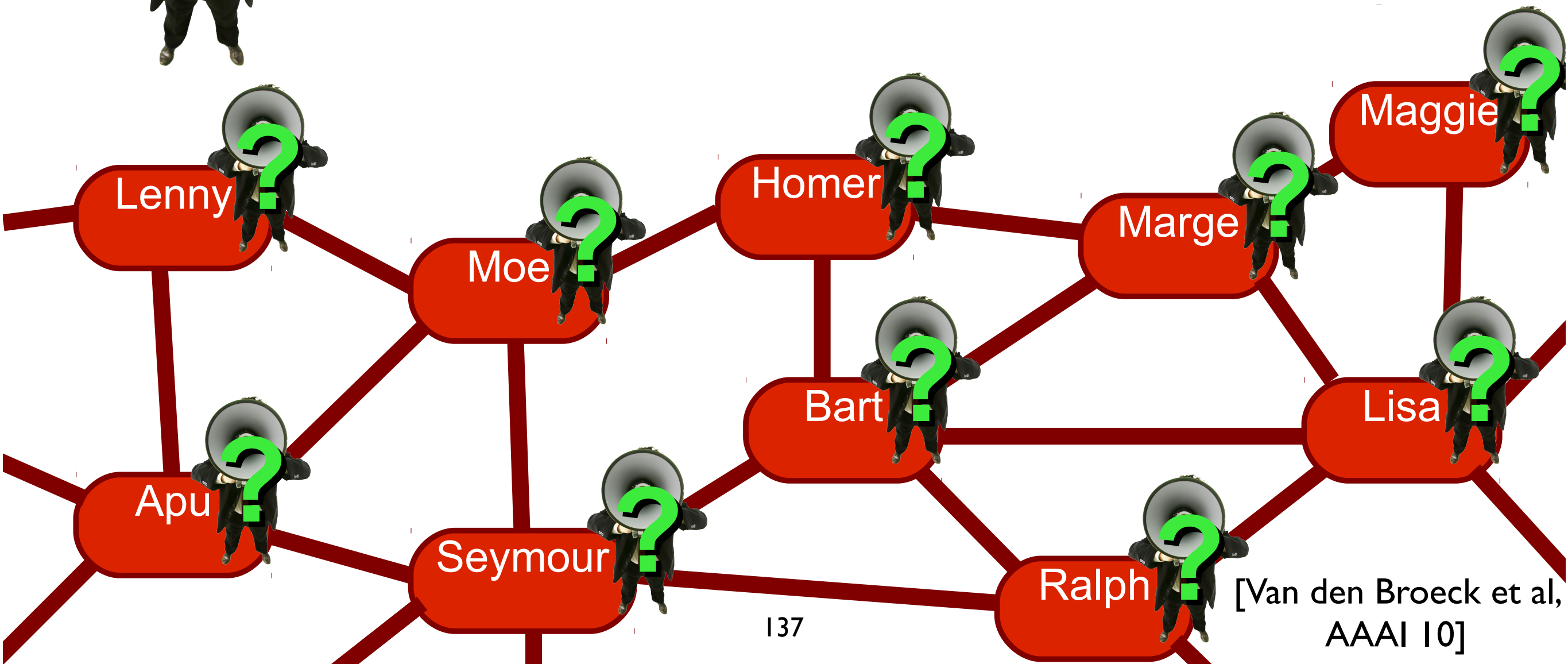
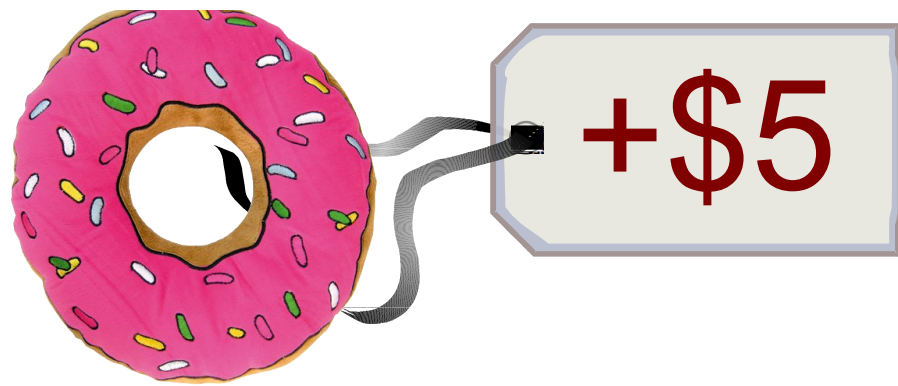
Event Calculus in ProbLog



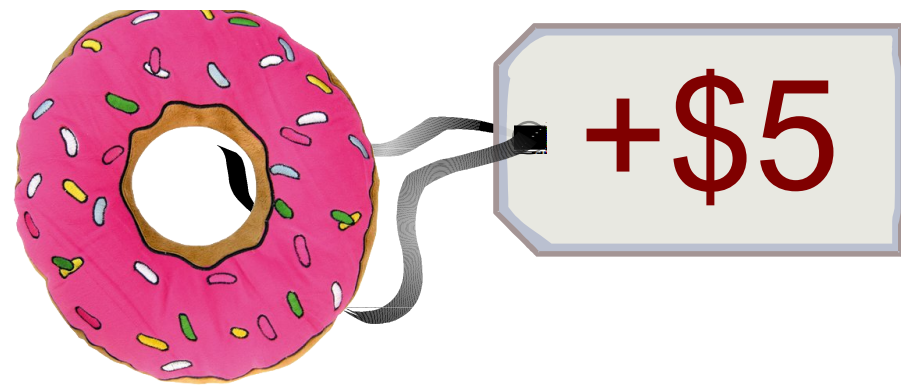
Decisions

Viral Marketing

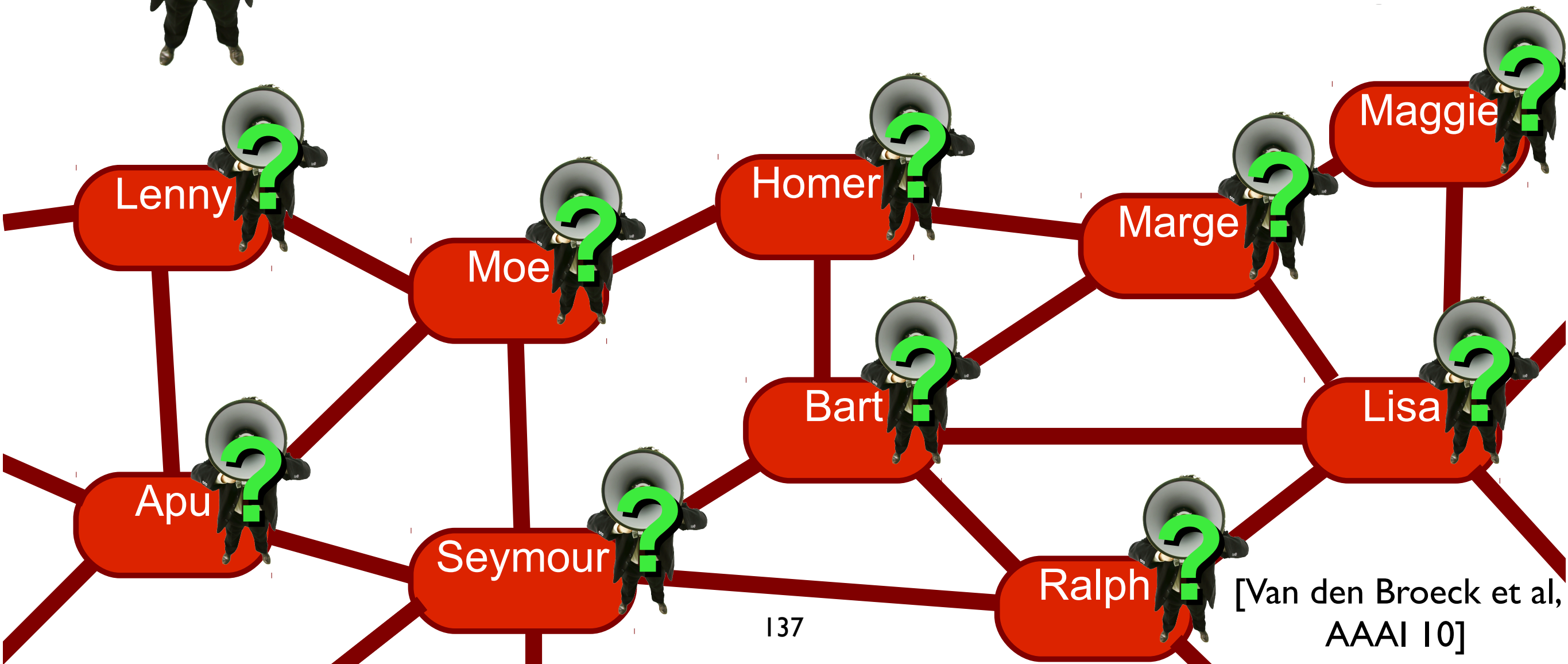
Which advertising strategy maximizes expected profit?



Viral Marketing

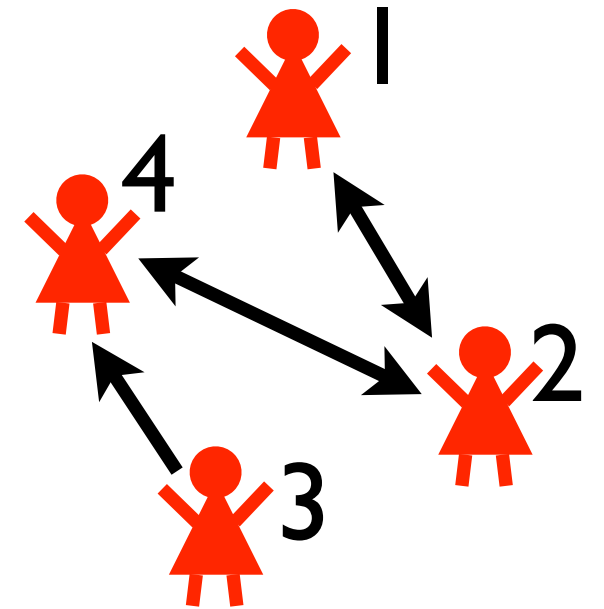


decide truth values of
some atoms



[Van den Broeck et al,
AAAI 10]

DTPProbLog



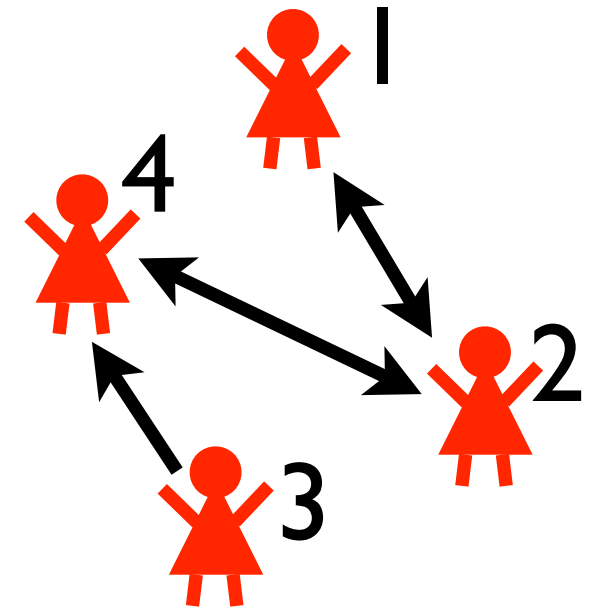
```
person(1).  
person(2).  
person(3).  
person(4).
```

```
friend(1,2).  
friend(2,1).  
friend(2,4).  
friend(3,4).  
friend(4,2).
```

DTPProbLog

`? :: marketed(P) :- person(P) .`

decision fact: true or false?



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

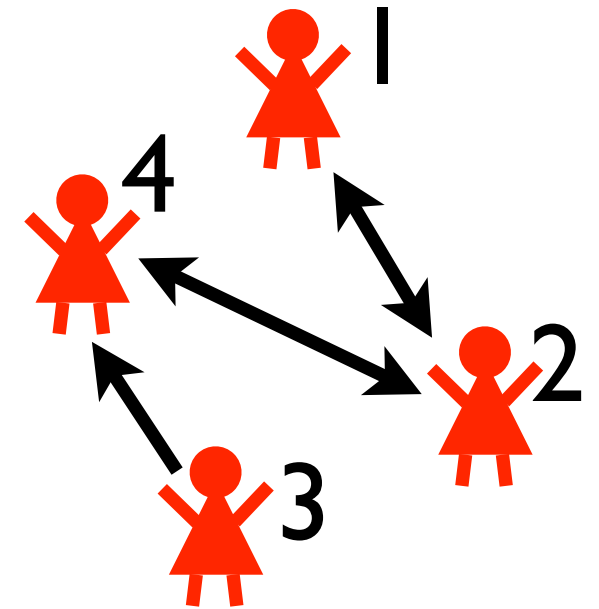
```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

**probabilistic facts
+ logical rules**



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

```
0.2 :: buy_marketing(P) :- person(P) .
```

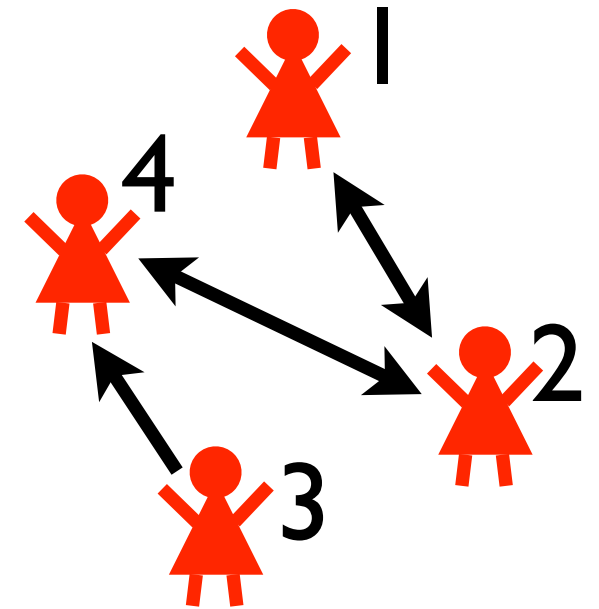
```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```

utility facts: cost/reward if true



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

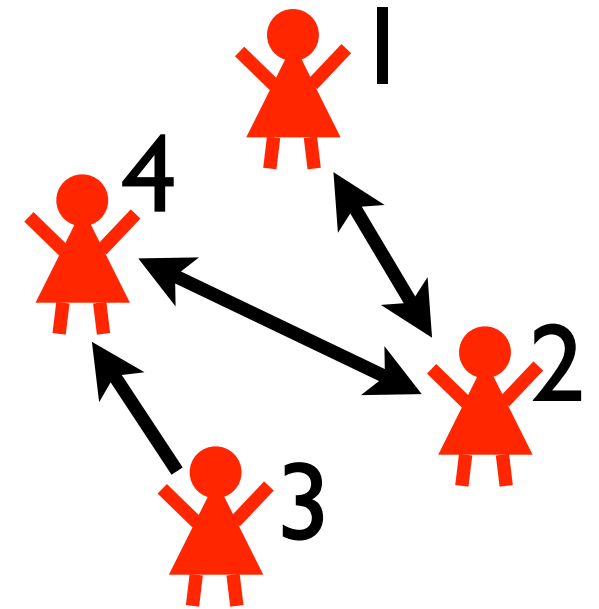
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

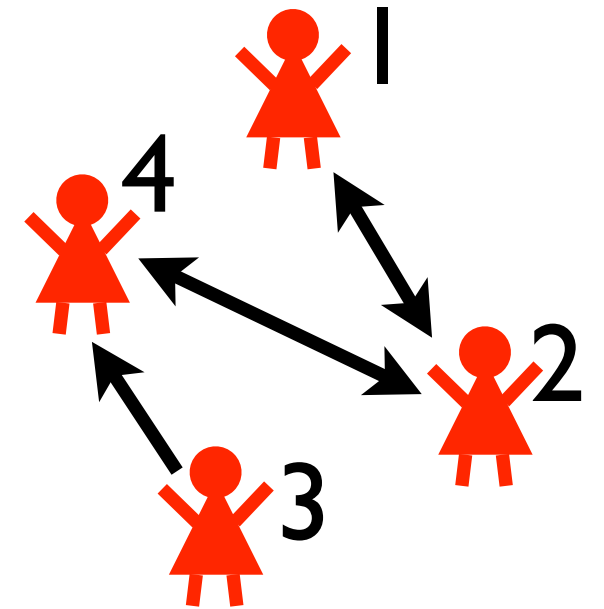
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

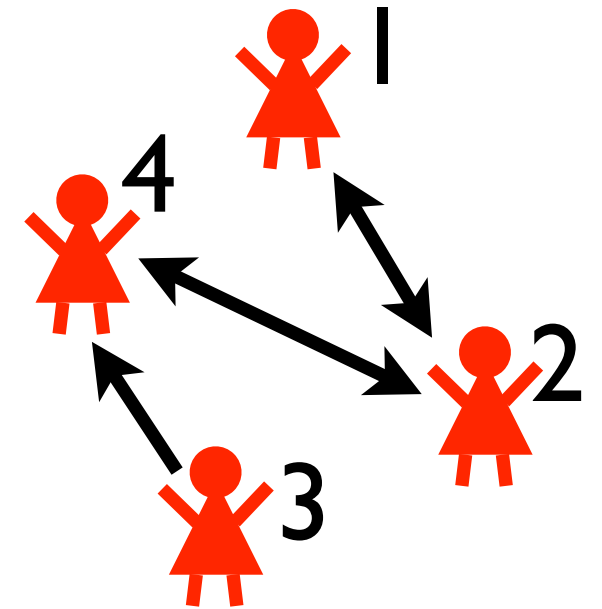
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

marketed(1)

marketed(3)

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

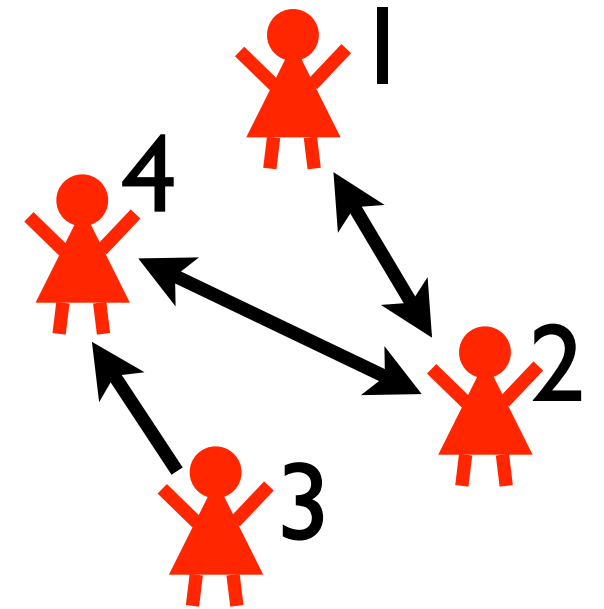
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

marketed(1)

marketed(3)

bt(2,1)

bt(2,4)

bm(1)

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

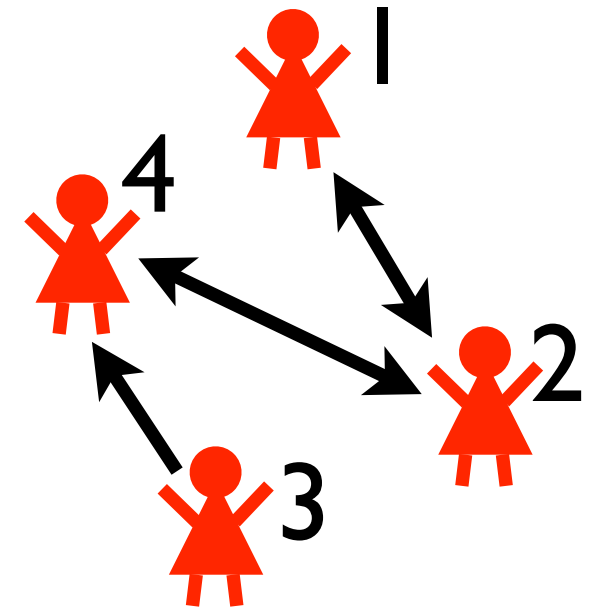
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .  
person(2) .  
person(3) .  
person(4) .
```

```
friend(1,2) .  
friend(2,1) .  
friend(2,4) .  
friend(3,4) .  
friend(4,2) .
```

marketed(1)	marketed(3)	
bt(2,1)	bt(2,4)	bm(1)
buys(1)	buys(2)	

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

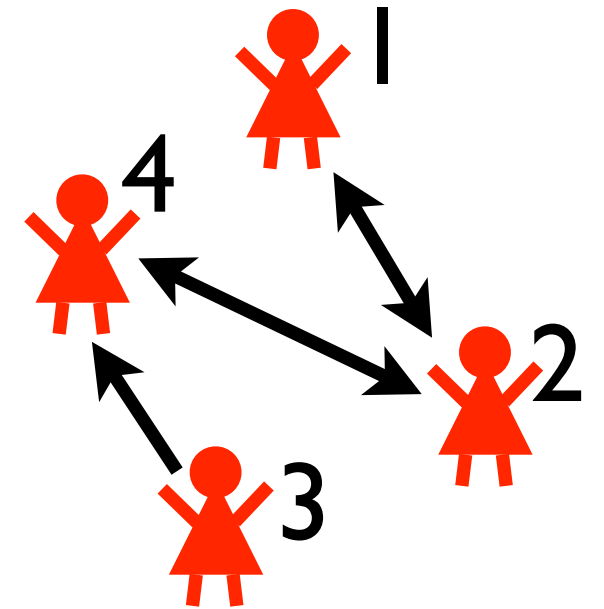
```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```

$$\text{utility} = -3 + -3 + 5 + 5 = 4$$

$$\text{probability} = 0.0032$$

marketed(1)	marketed(3)	
bt(2,1)	bt(2,4)	bm(1)
buys(1)	buys(2)	



```
person(1) .
person(2) .
person(3) .
person(4) .
```

```
friend(1,2) .
friend(2,1) .
friend(2,4) .
friend(3,4) .
friend(4,2) .
```

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```

$$\text{utility} = -3 + -3 + 5 + 5 = 4$$

$$\text{probability} = 0.0032$$

marketed(1)

marketed(3)

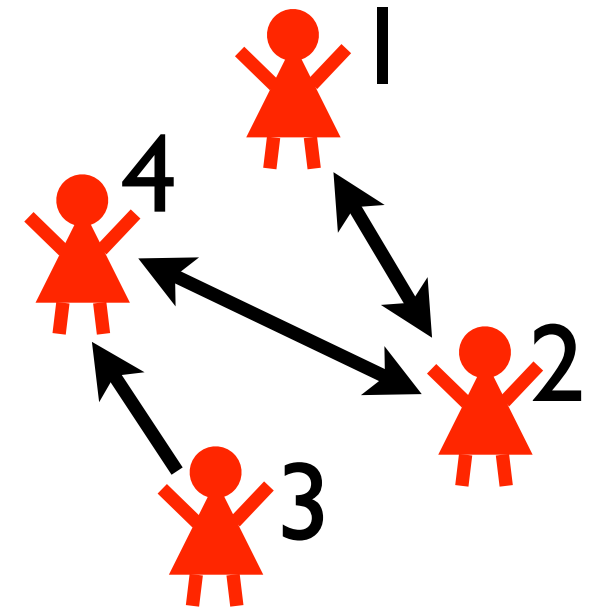
bt(2,1)

bt(2,4)

bm(1)

buys(1)

buys(2)



person(1) .

person(2) .

person(3) .

person(4) .

friend(1,2) .

friend(2,1) .

friend(2,4) .

friend(3,4) .

friend(4,2) .

world contributes

0.0032×4 to

expected utility of
strategy

DTPProbLog

```
? :: marketed(P) :- person(P) .
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y) .
```

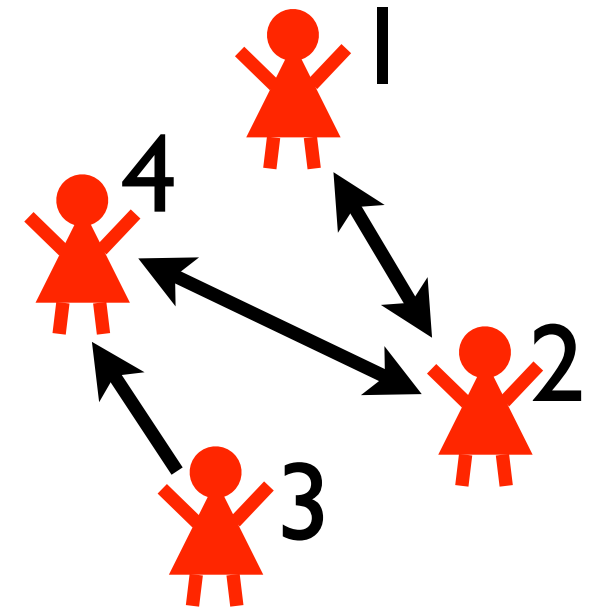
```
0.2 :: buy_marketing(P) :- person(P) .
```

```
buys(X) :- friend(X,Y) , buys(Y) , buy_trust(X,Y) .
```

```
buys(X) :- marketed(X) , buy_marketing(X) .
```

```
buys(P) => 5 :- person(P) .
```

```
marketed(P) => -3 :- person(P) .
```



```
person(1) .
```

```
person(2) .
```

```
person(3) .
```

```
person(4) .
```

```
friend(1,2) .
```

```
friend(2,1) .
```

```
friend(2,4) .
```

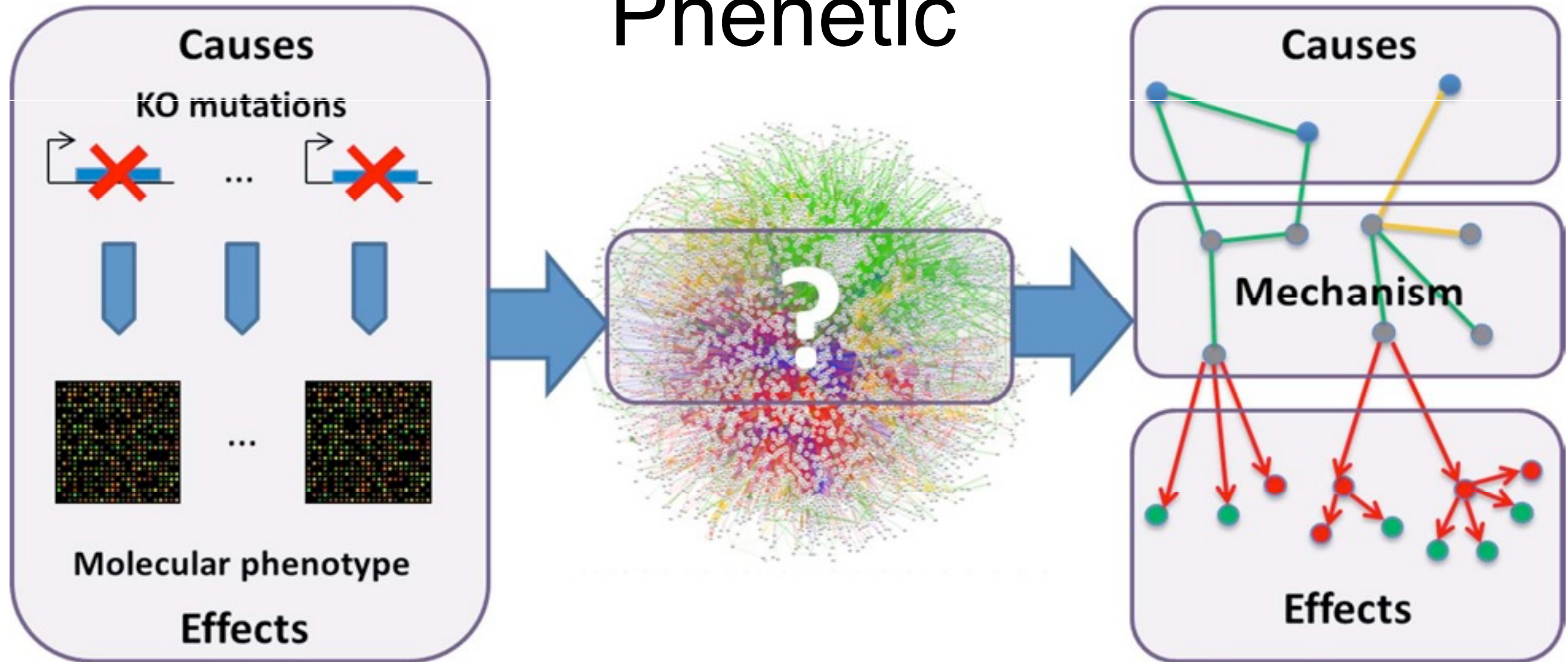
```
friend(3,4) .
```

```
friend(4,2) .
```

task: find strategy that maximizes expected utility

solution: using ProbLog technology

Phenetic



- Causes: Mutations
 - All related to similar phenotype
- Effects: Differentially expressed genes
 - 27 000 cause effect pairs

- Interaction network:
 - 3063 nodes
 - Genes
 - Proteins
 - 16794 edges
 - Molecular interactions
 - Uncertain

- Goal: connect causes to effects through common subnetwork
 - = Find mechanism
- Techniques:
 - DTProbLog
 - Approximate inference

Roadmap

- Modeling
- Reasoning
- Language extensions
- Advanced topics

... with some detours on the way

Advanced Topics

- learning
- complexity of querying
- lifted graphical models and KBMC

Learning

Parameter Learning

e.g., webpage classification model

for each **CLASS1**, **CLASS2** and each **WORD**

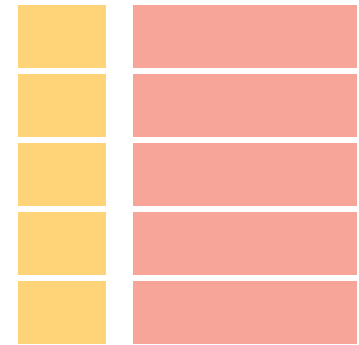
```
?? :: link_class(Source,Target,CLASS1,CLASS2).
```

```
?? :: word_class(WORD,CLASS).
```

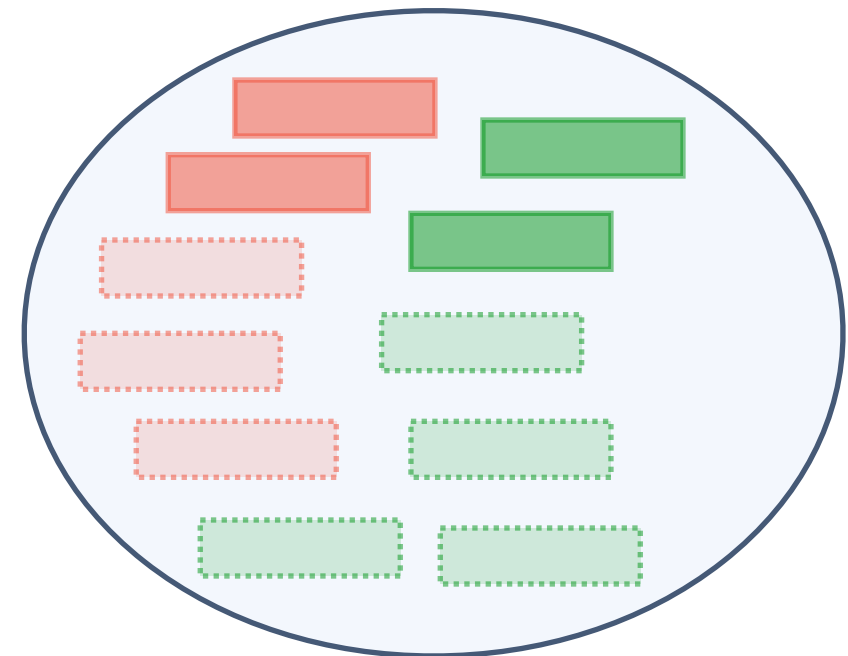
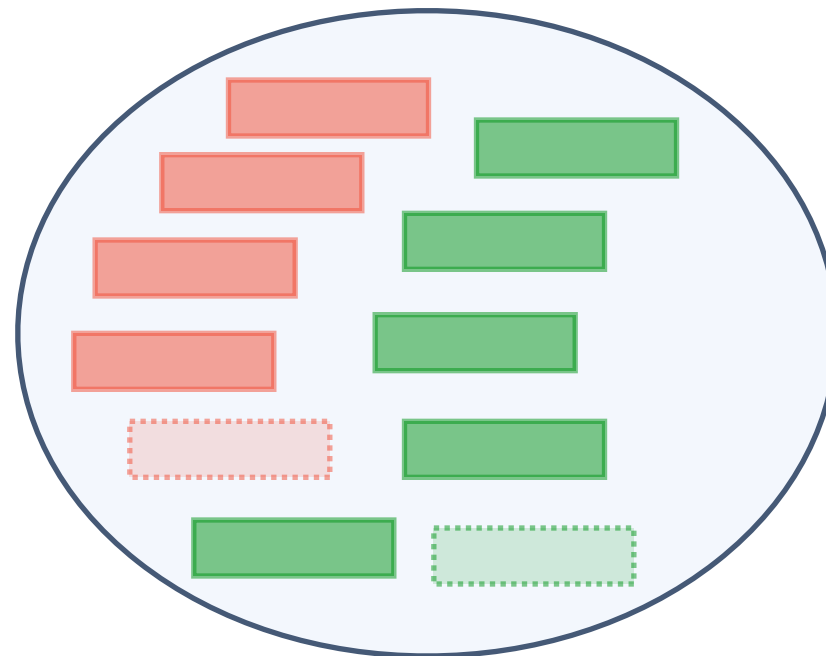
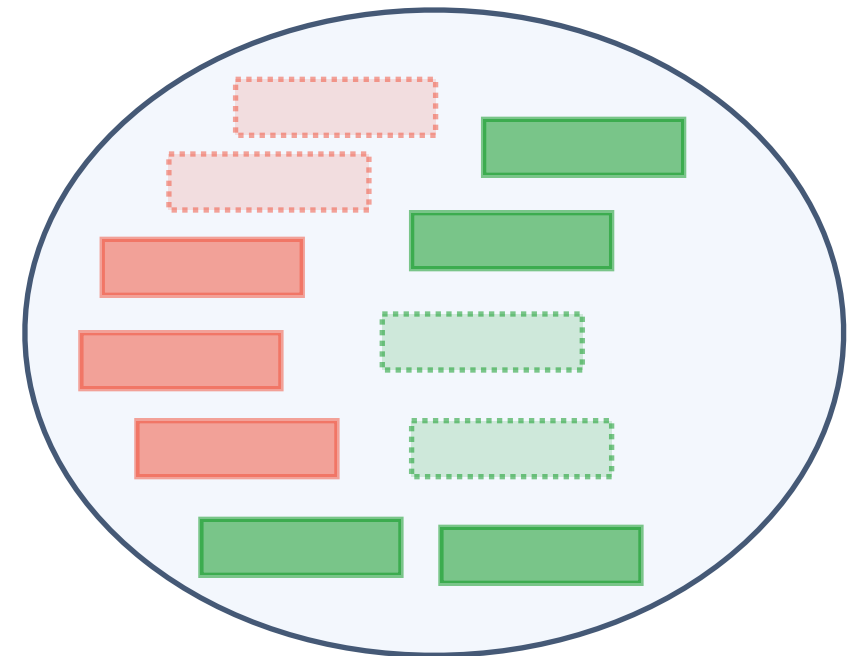
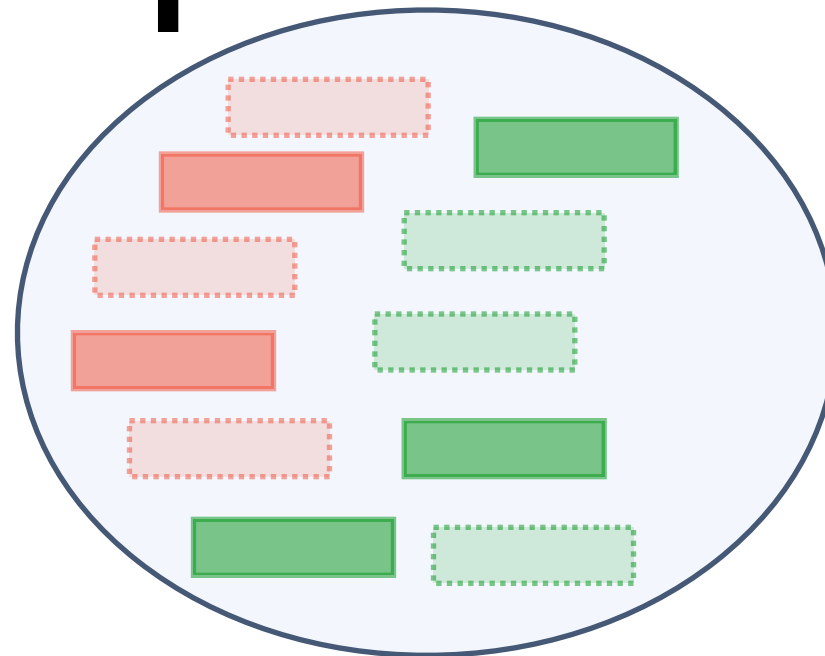
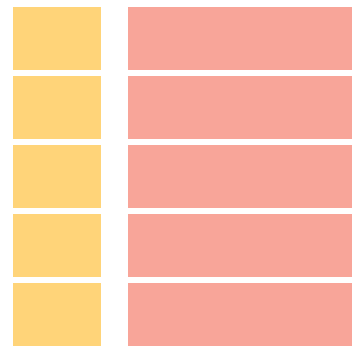
```
class(Page,C) :- has_word(Page,W), word_class(W,C).
```

```
class(Page,C) :- links_to(OtherPage,Page),  
                  class(OtherPage,OtherClass),  
                  link_class(OtherPage,Page,OtherClass,C).
```

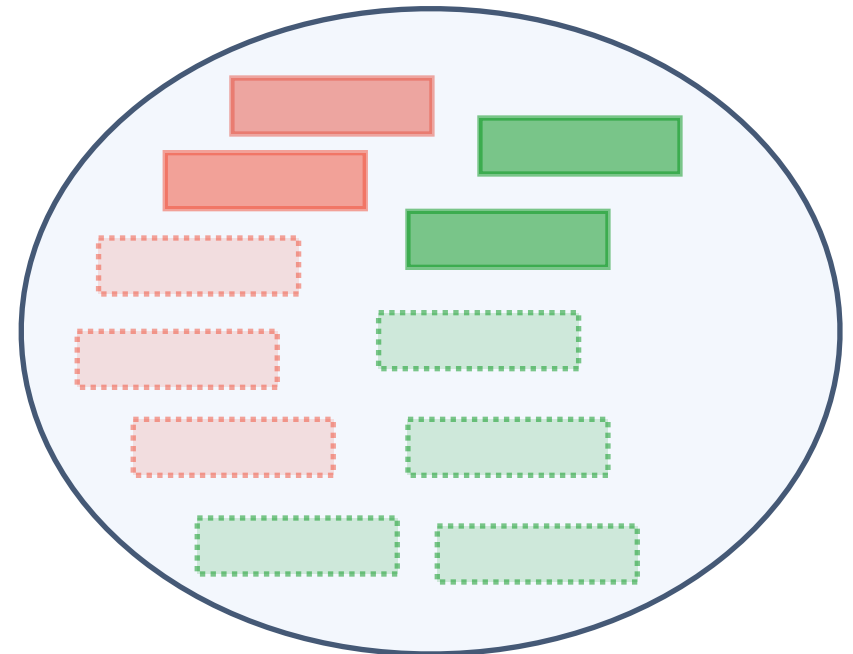
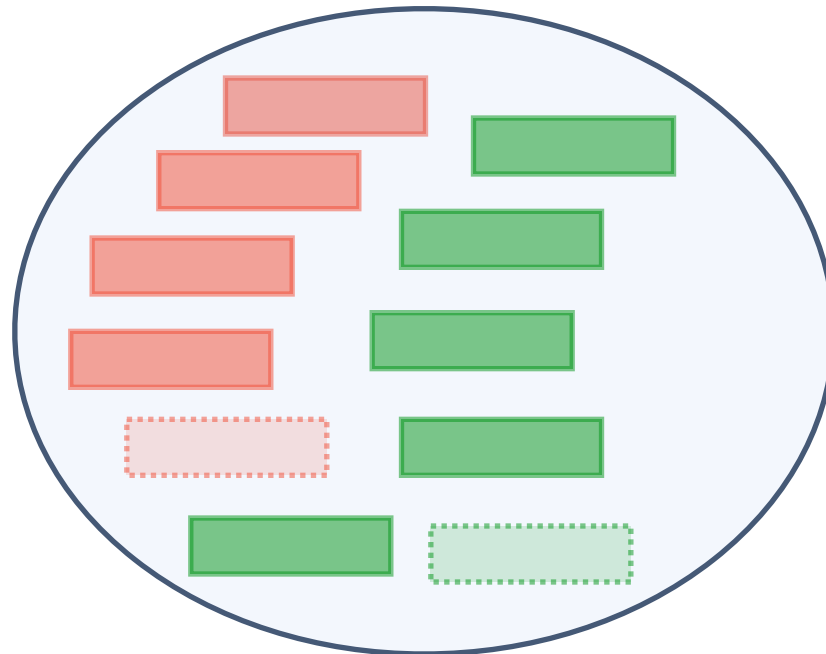
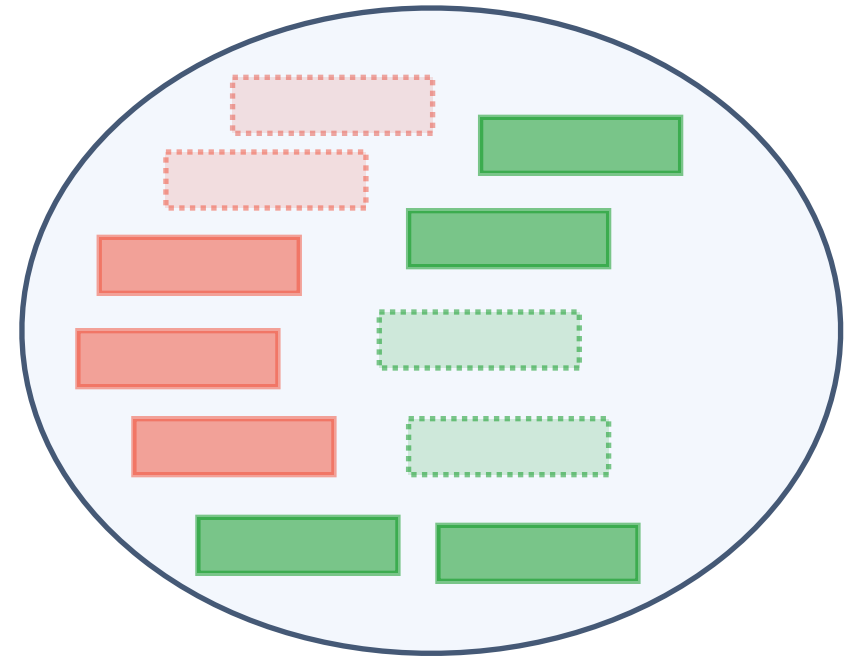
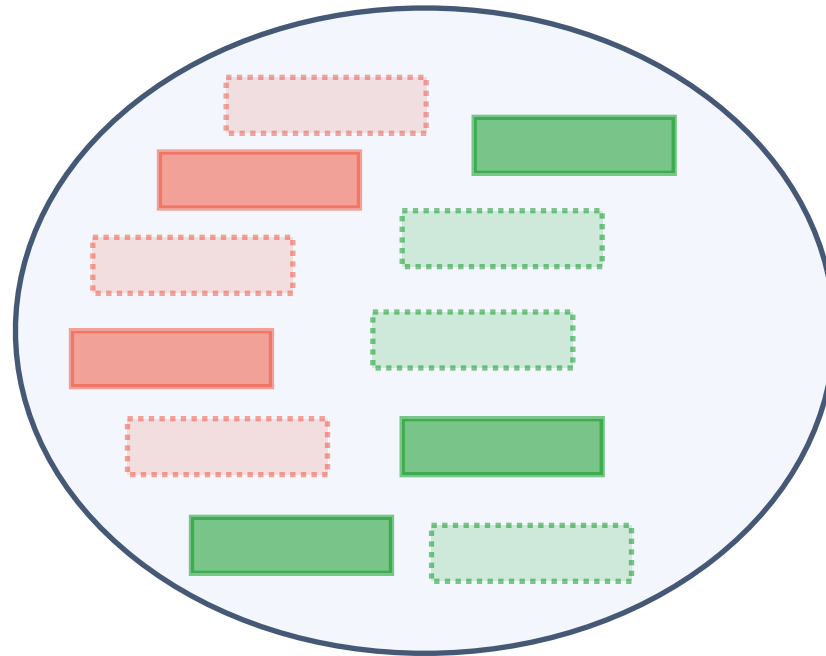
Sampling Interpretations



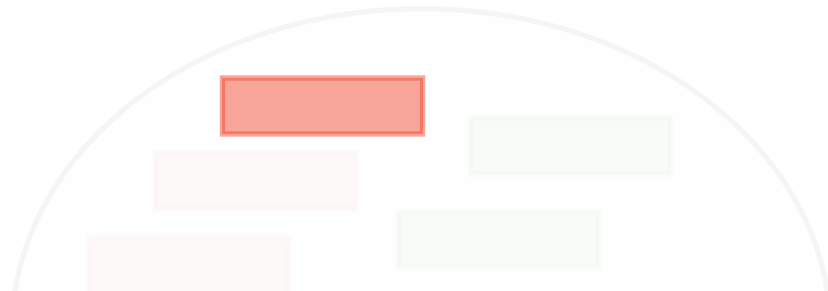
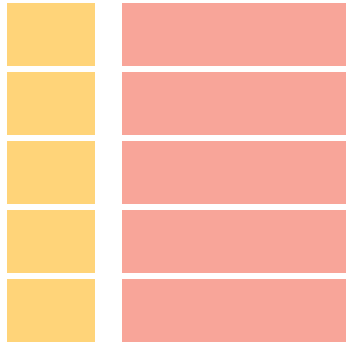
Sampling Interpretations



Parameter Estimation

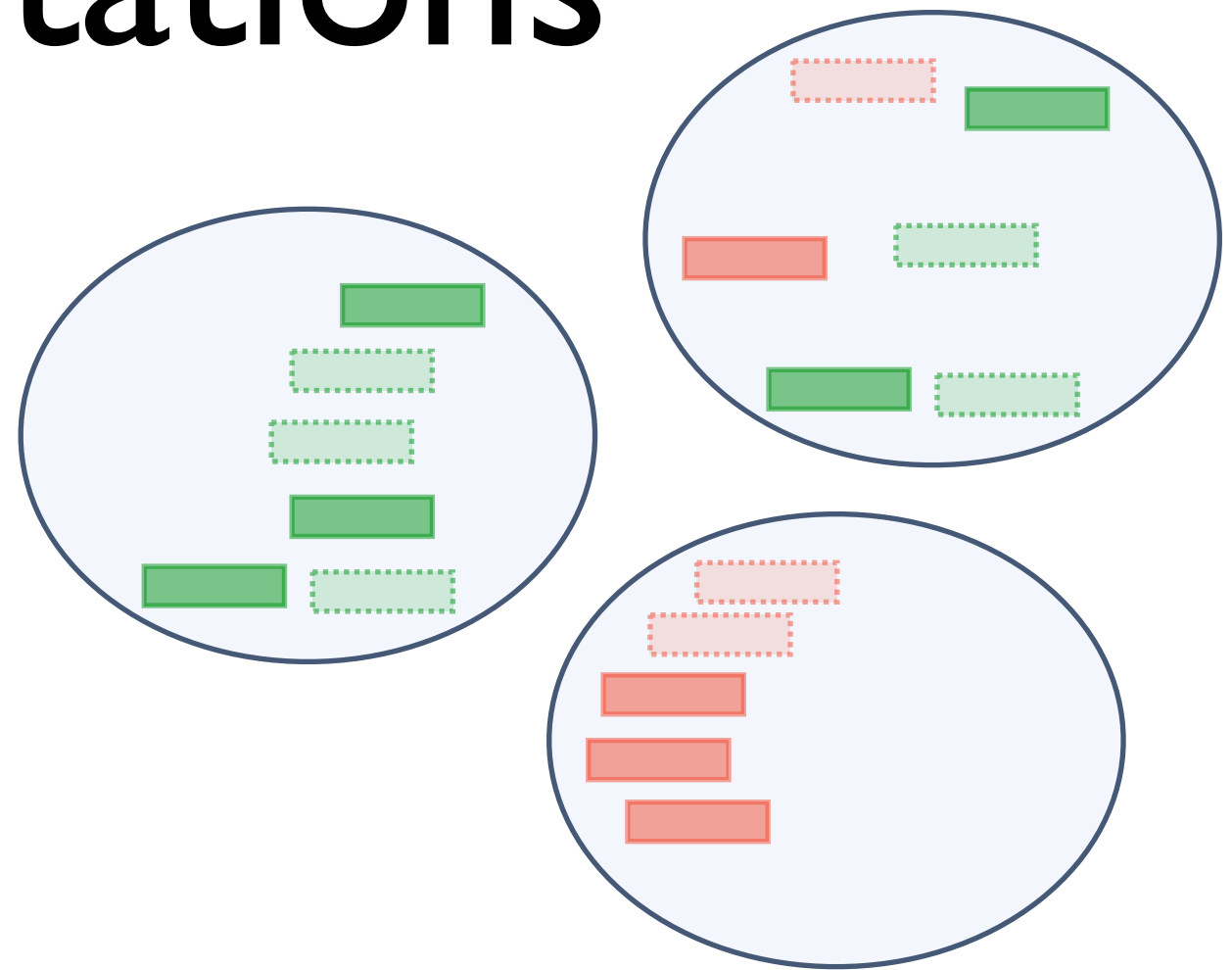


Parameter Estimation



$$p(\mathbf{fact}) = \frac{\text{count}(\mathbf{fact} \text{ is true})}{\text{Number of interpretations}}$$

Learning from partial interpretations



- Not all facts observed
- Soft-EM
- use **expected count** instead of **count**
- **$P(Q | E)$ -- conditional queries !**

Bayesian Parameter Learning

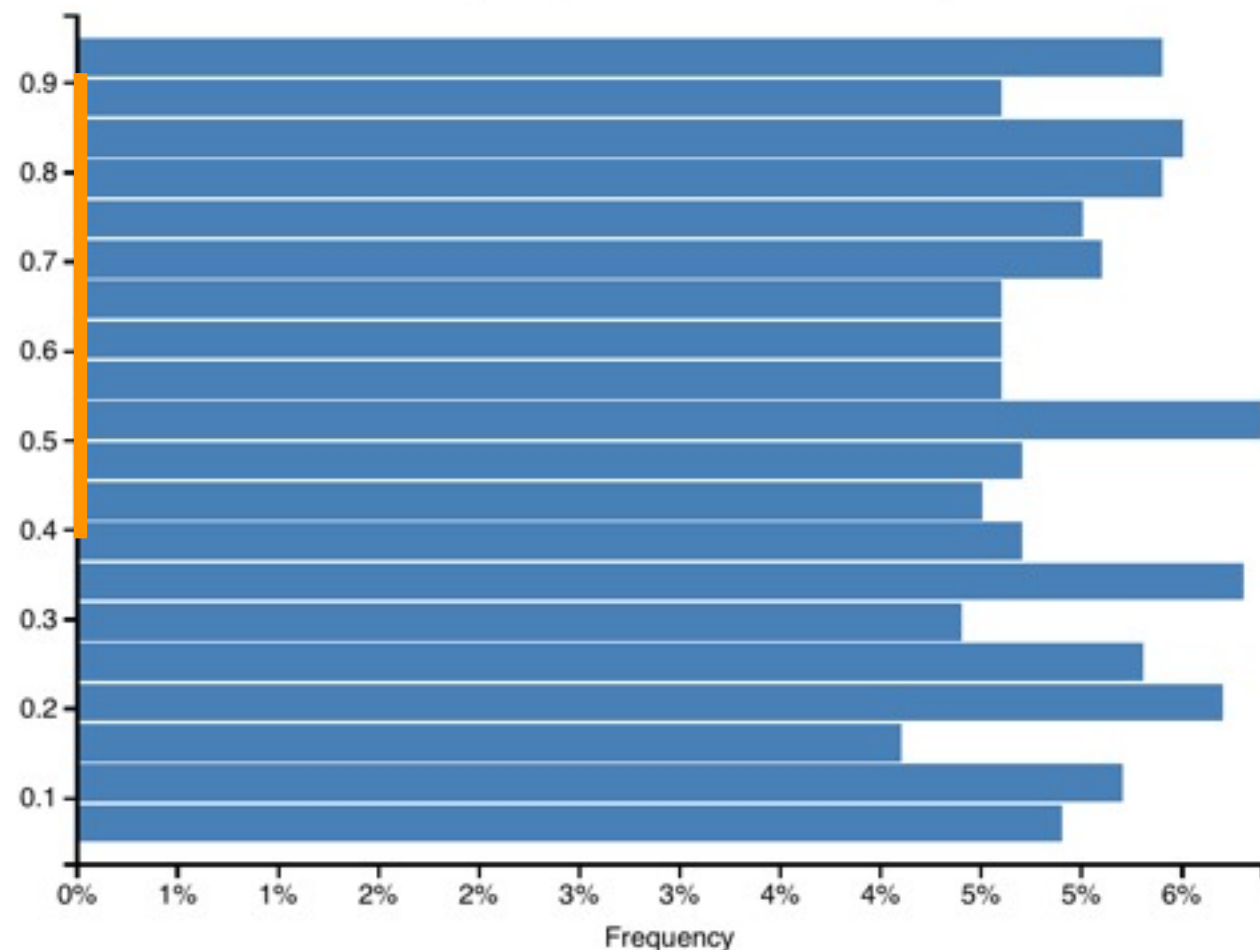
- Learning as inference (e.g., Church)
- Prior distributions for parameters
- Given data, find most likely parameter values

Example

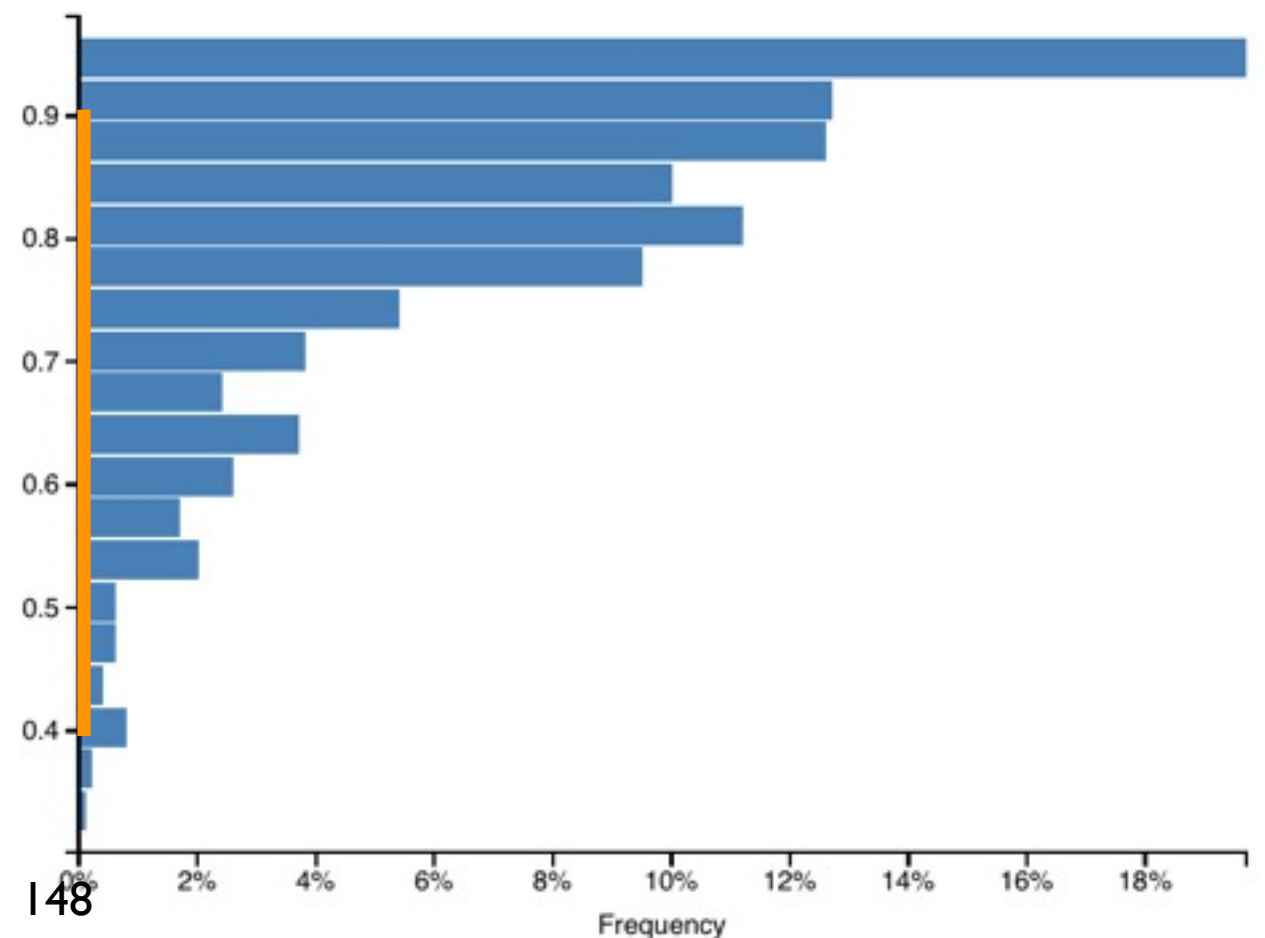
[from probmods.org]

- Flipping a coin with unknown weight
- Prior: uniform distribution on $[0, 1]$
- Observation: 5x heads in a row
- Sampling from Church model:

Coin weight, prior to observing data



Coin weight, conditioned on observed data



ProbLog Example

prior

```
0.05::weight(C,0.1) ; 0.2::weight(C,0.3) ; 0.5::weight(C,0.5) ;  
0.2::weight(C,0.7) ; 0.05::weight(C,0.9) <- coin(C) .
```

```
Param::toss(_,Param,_).  
heads(C,R) :- weight(C,Param),toss(C,Param,R).  
tails(C,R) :- weight(C,Param),\+toss(C,Param,R).  
  
data(C,[]).  
data(C,[h|R]) :- heads(C,R), data(C,R).  
data(C,[t|R]) :- tails(C,R), data(C,R).  
  
coin(c1).  
coin(c2).  
param(0.1).  
param(0.3).  
param(0.5).  
param(0.7).  
param(0.9).
```

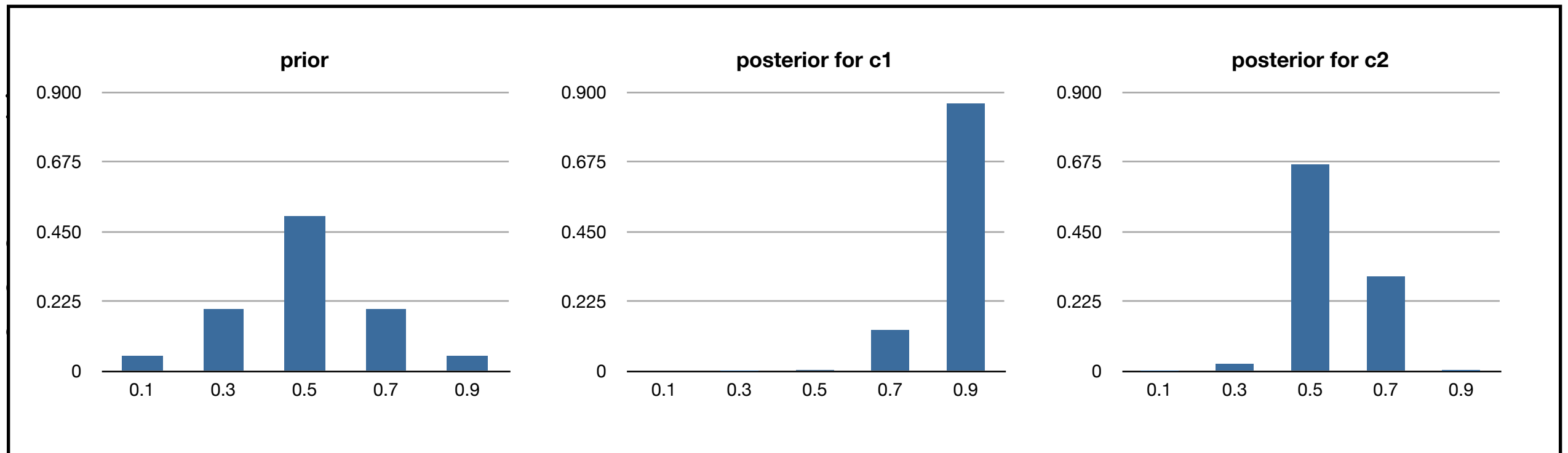
```
query(weight(C,X)) :- coin(C),param(X). ask for posterior
```

```
evidence(data(c1,[h,h,h,h,h,h,h,h,h,h,h,h,h,h]),true).  
evidence(data(c2,[h,t,h,h,h,h,h,t,t,h,t,t,h]),true).
```

data

ProbLog Example

```
0.05::weight(C,0.1) ; 0.2::weight(C,0.3) ; 0.5::weight(C,0.5) ;  
0.2::weight(C,0.7) ; 0.05::weight(C,0.9) <- coin(C) .
```



```
query(weight(C,X)) :- coin(C),param(X) .
```

```
evidence(data(c1,[h,h,h,h,h,h,h,h,h,h,h,h,h,h]),true) .
```

```
evidence(data(c2,[h,t,h,h,h,h,h,t,t,h,t,t,h]),true) .
```

Upgrading relational learning

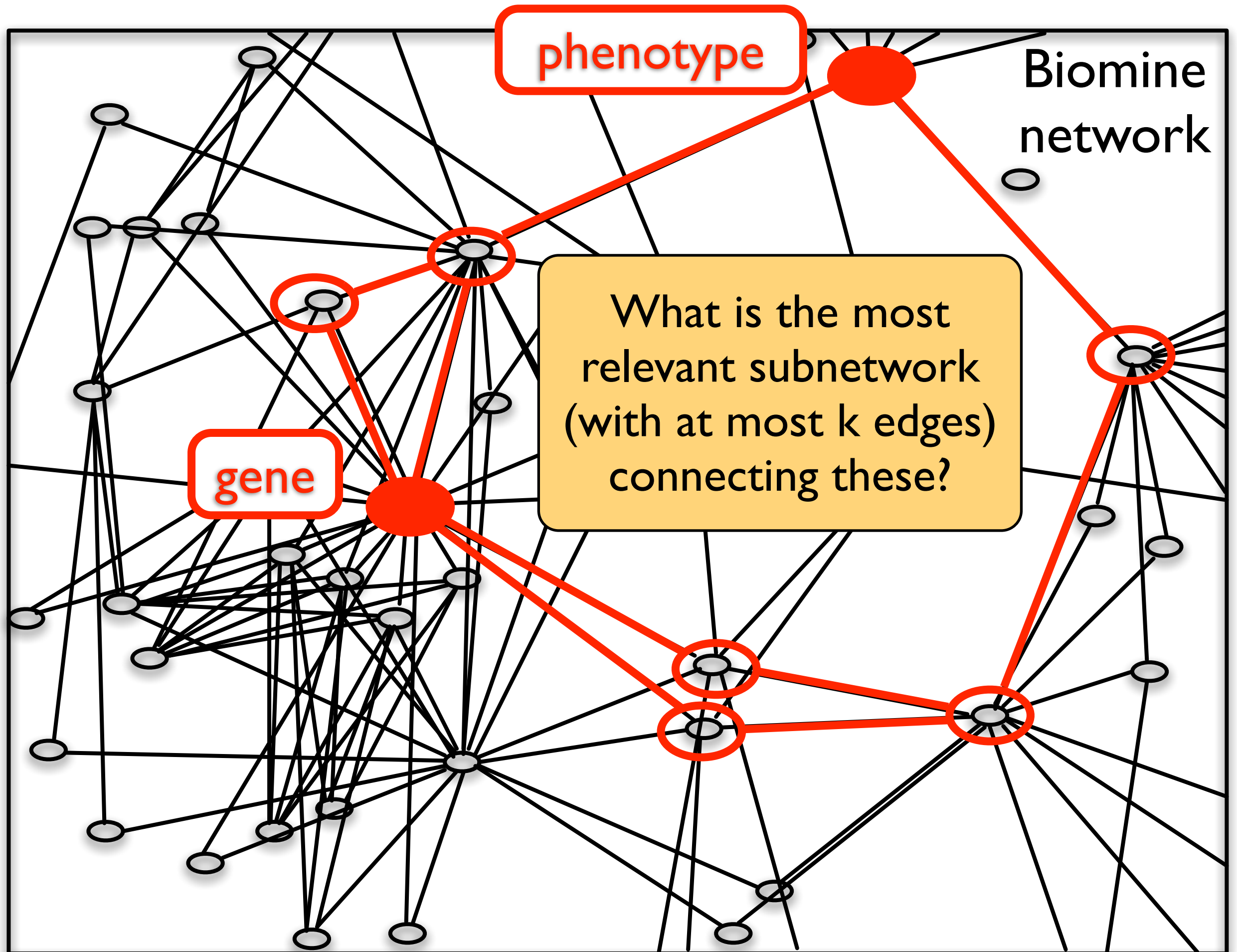
	Prolog	ProbLog
	<code>infl(a,b) .</code> ...	<code>0.4::infl(a,b) .</code> ...
Reasoning	query true? yes/no	query true? with probability P

Upgrading relational learning

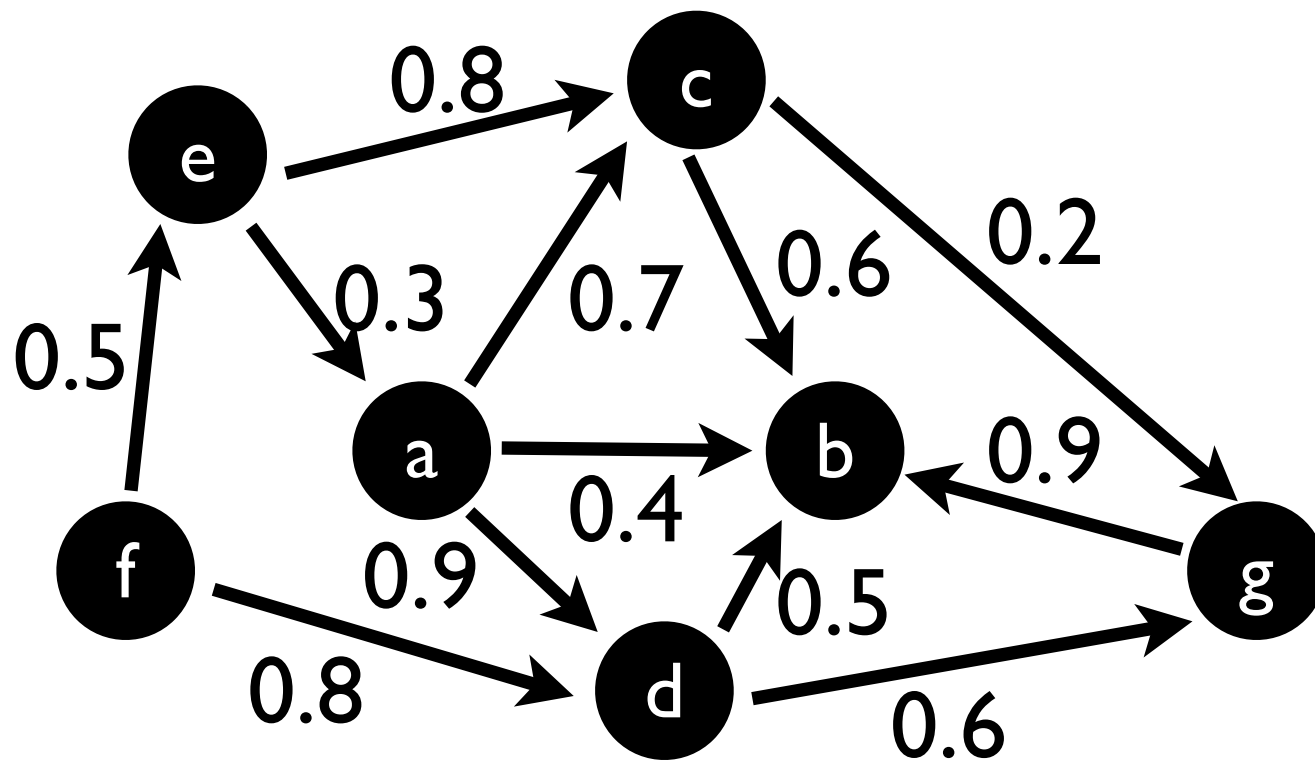
	Prolog	ProbLog
	<code>infl(a,b) .</code> ...	<code>0.4::infl(a,b) .</code> ...
Reasoning	query true? yes/no	query true? with probability P
Machine Learning	example covered? yes/no	example covered? with probability P

Structure Learning for LPADs

- Find the model and the parameters that maximize the probability of the data (log-likelihood)
- SLIPCAS: Structure Learning of Probabilistic logic programs with EM over bdds [Bellodi & Riguzzi IJLP 11]
- Beam search in the space of probabilistic programs
- SLIPCOVER: Structure Learning of Probabilistic logic program by searching OVER the clause space [Bellodi & Riguzzi TPLP 14]
- Beam search in the space of clauses to find the promising ones
- Greedy search in the space of probabilistic programs guided by the LL of the data.
- Both perform parameter learning by means of EMBLEM



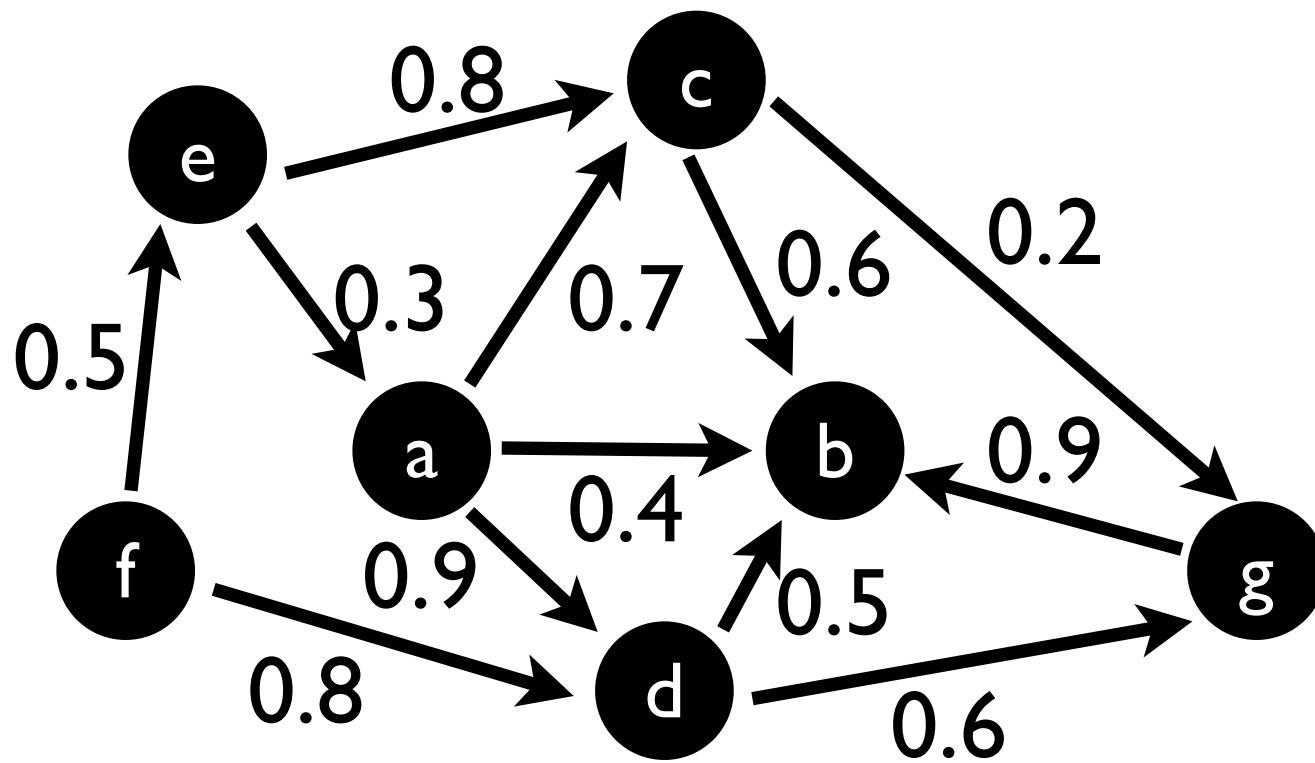
Theory compression



```
path(X,Y) :- edge(X,Y) .  
path(X,Y) :- edge(X,Z) , path(Z,Y) .
```

```
0.8::edge(e,c) .  
0.3::edge(e,a) .  
...
```

Theory compression

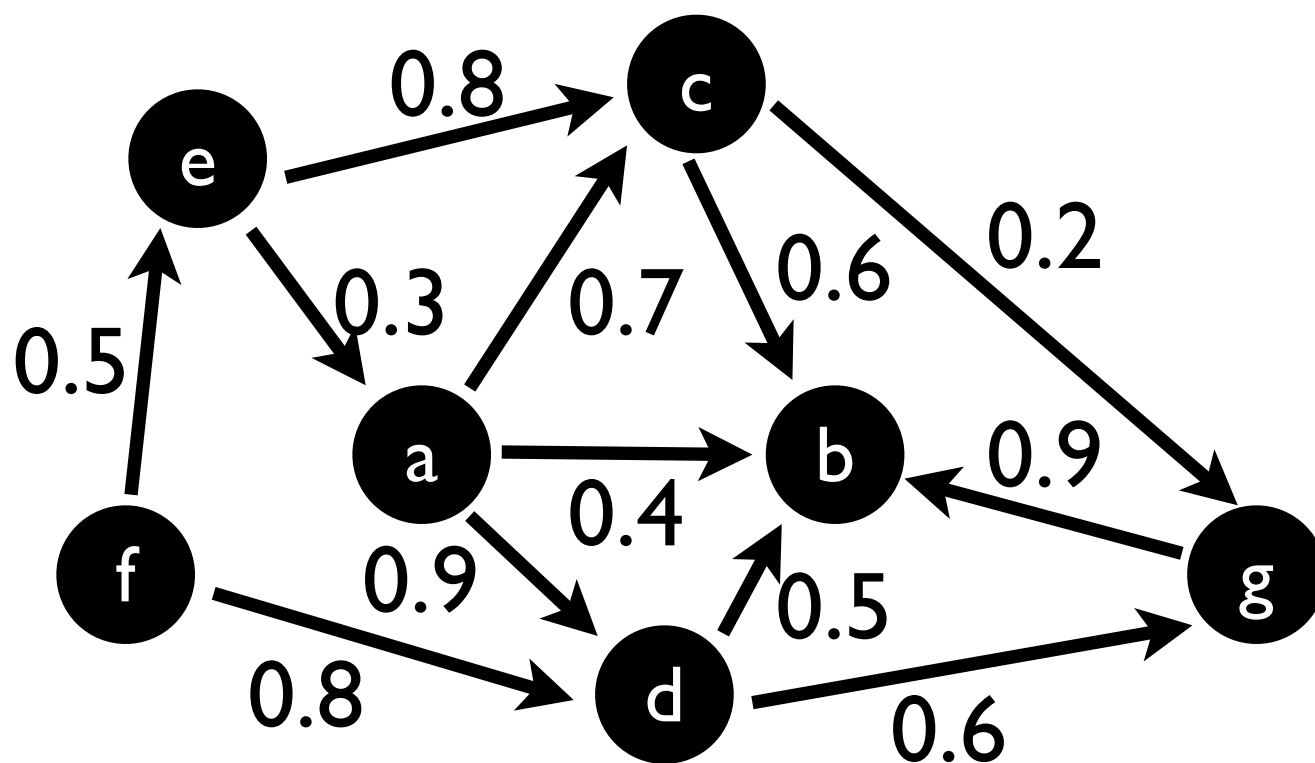


```
path(X,Y) :- edge(X,Y) .  
path(X,Y) :- edge(X,Z) , path(Z,Y) .
```

```
0.8::edge(e,c) .  
0.3::edge(e,a) .  
...
```

best subnetwork of at most
5 edges where **path(a,b)**
but not **path(e,g)**?

Theory compression



```

path(X,Y) :- edge(X,Y) .
path(X,Y) :- edge(X,Z) , path(Z,Y) .

```

```

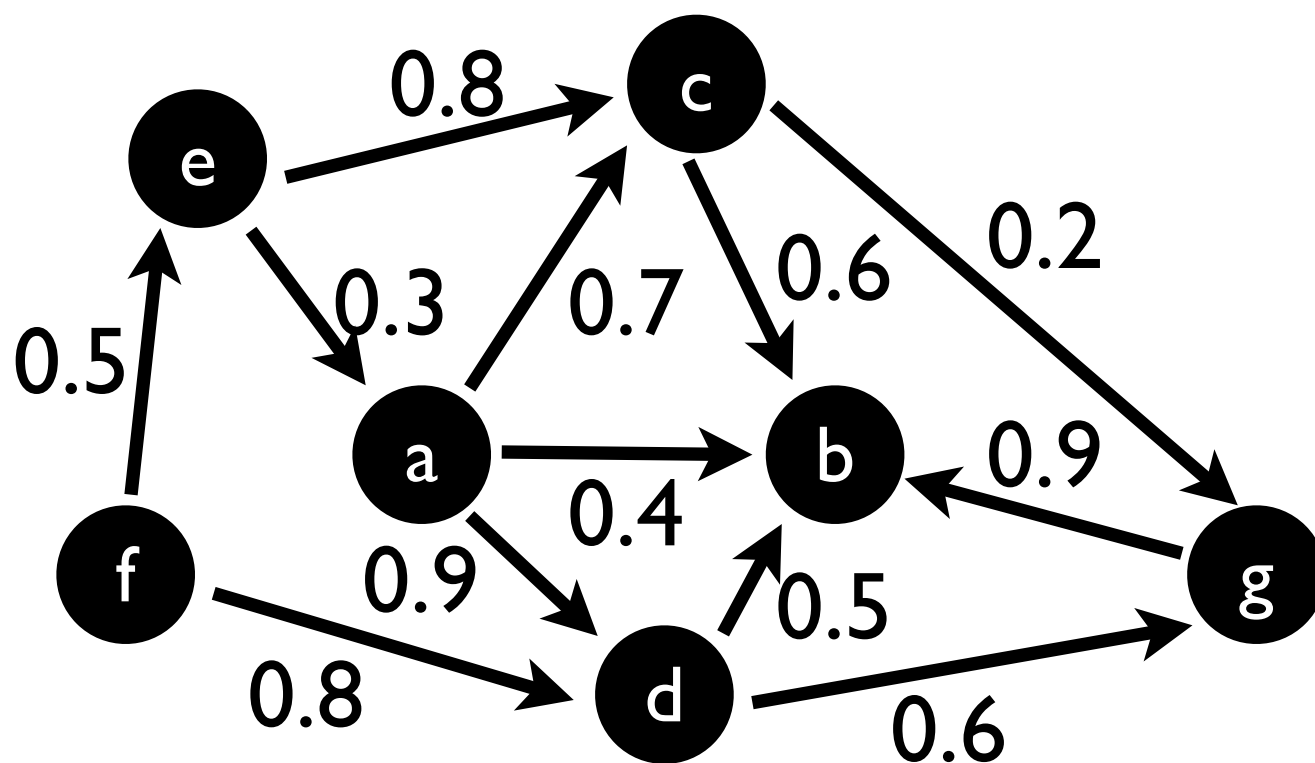
0.8 :: edge(e,c) .
0.3 :: edge(e,a) .
...

```

best subnetwork of at most
5 edges where **path(a,b)**
but not **path(e,g)**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$

Theory compression



```

path(X,Y) :- edge(X,Y) .
path(X,Y) :- edge(X,Z) , path(Z,Y) .

```

```

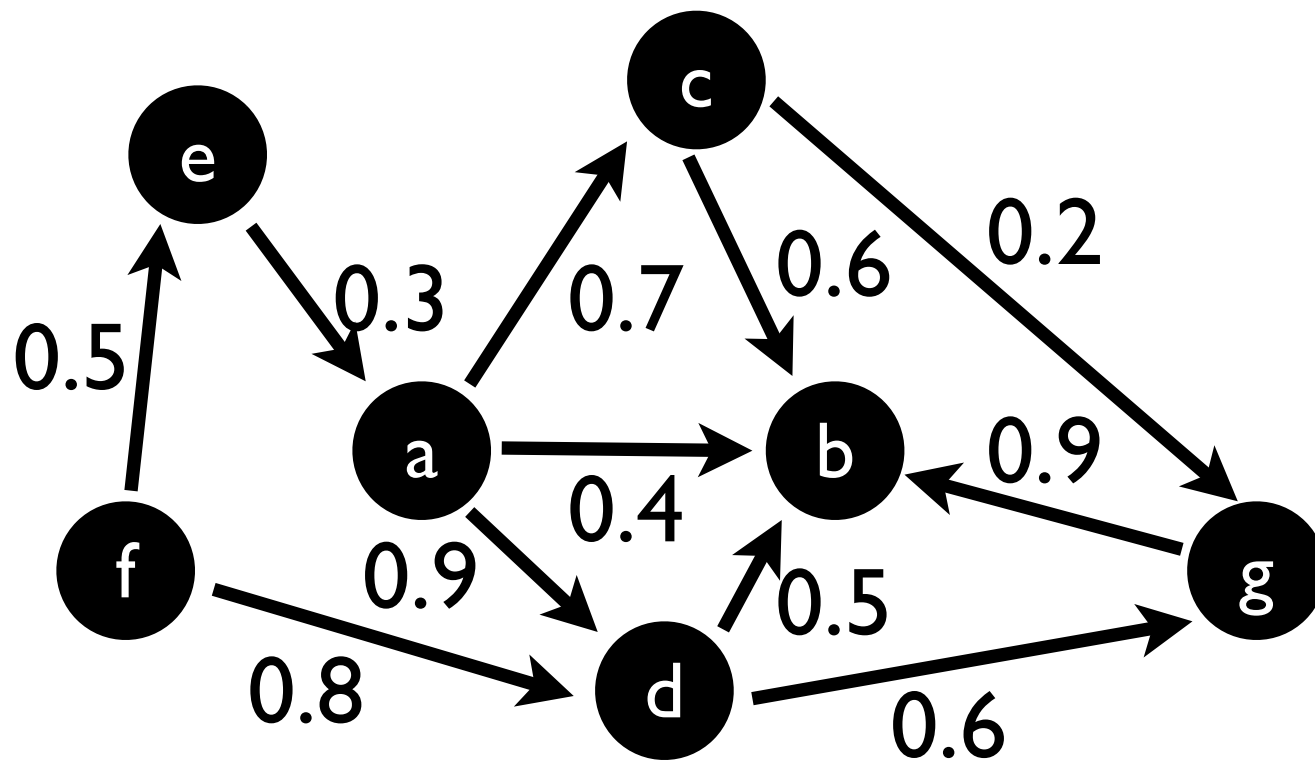
0.8 :: edge(e,c) .
0.3 :: edge(e,a) .
...

```

best subnetwork of at most
5 edges where **path(a,b)**
but not **path(e,g)**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$

Theory compression



```

path(X,Y) :- edge(X,Y) .
path(X,Y) :- edge(X,Z) , path(Z,Y) .

```

```

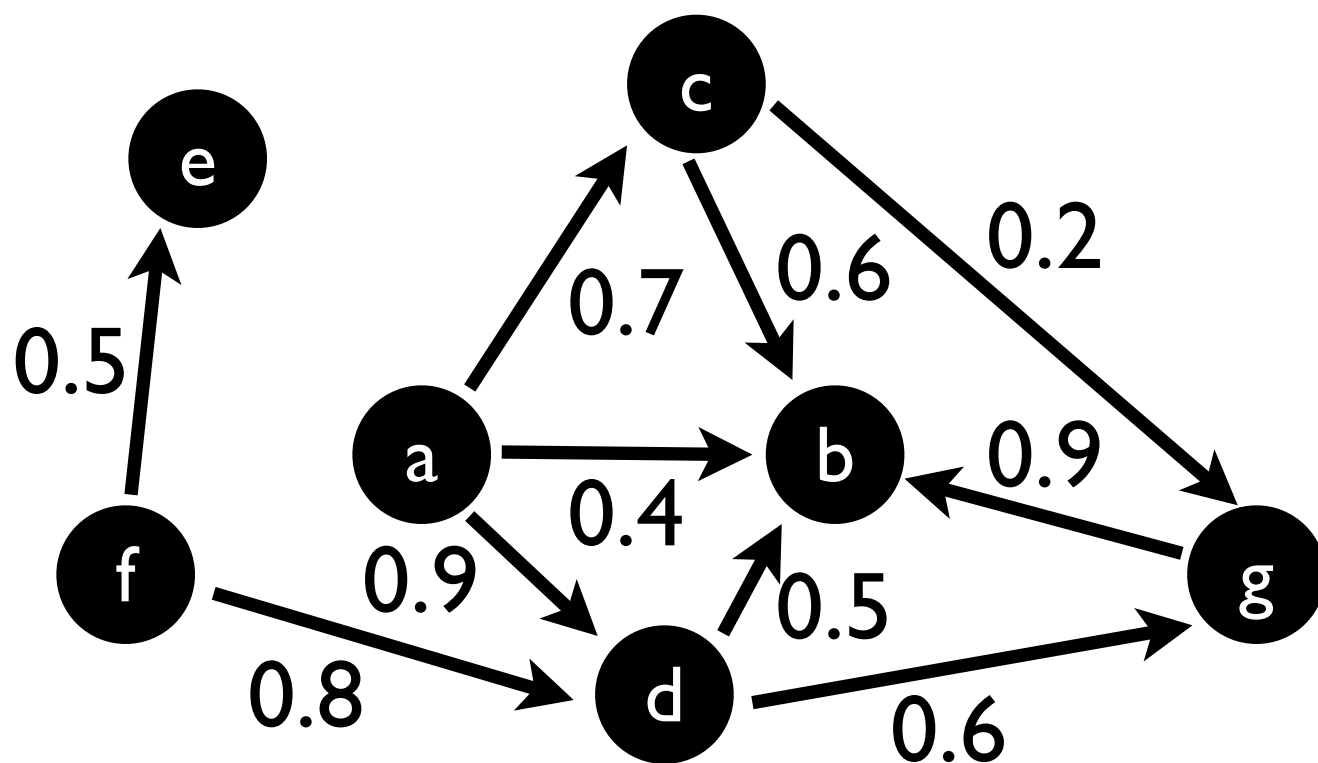
0.8::edge(e,c) .
0.3::edge(e,a) .
...

```

best subnetwork of at most
5 edges where **path(a,b)**
but not **path(e,g)**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$

Theory compression



```

path(X,Y) :- edge(X,Y) .
path(X,Y) :- edge(X,Z) , path(Z,Y) .

```

```

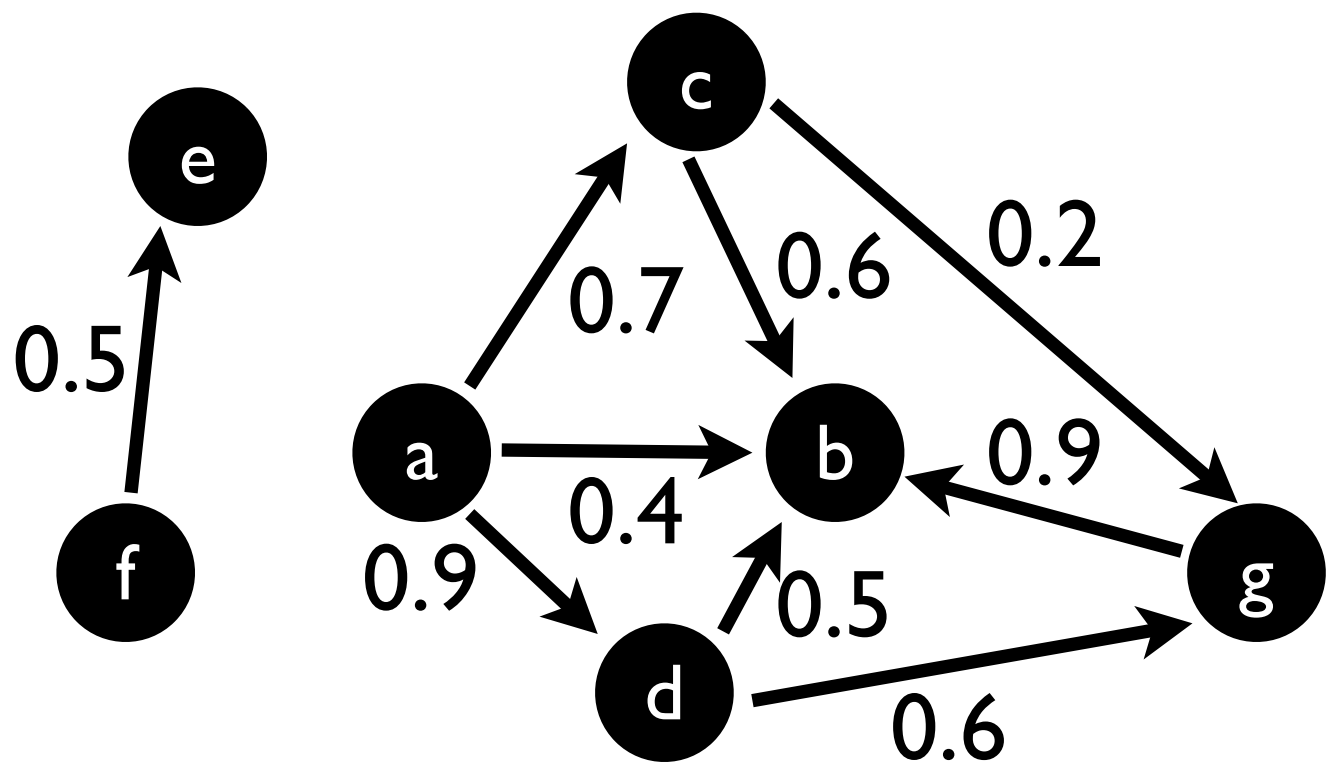
0.8::edge(e,c) .
0.3::edge(e,a) .
...

```

best subnetwork of at most
5 edges where **path(a,b)**
but not **path(e,g)**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$

Theory compression



```

path(X,Y) :- edge(X,Y) .
path(X,Y) :- edge(X,Z) , path(Z,Y) .

```

```

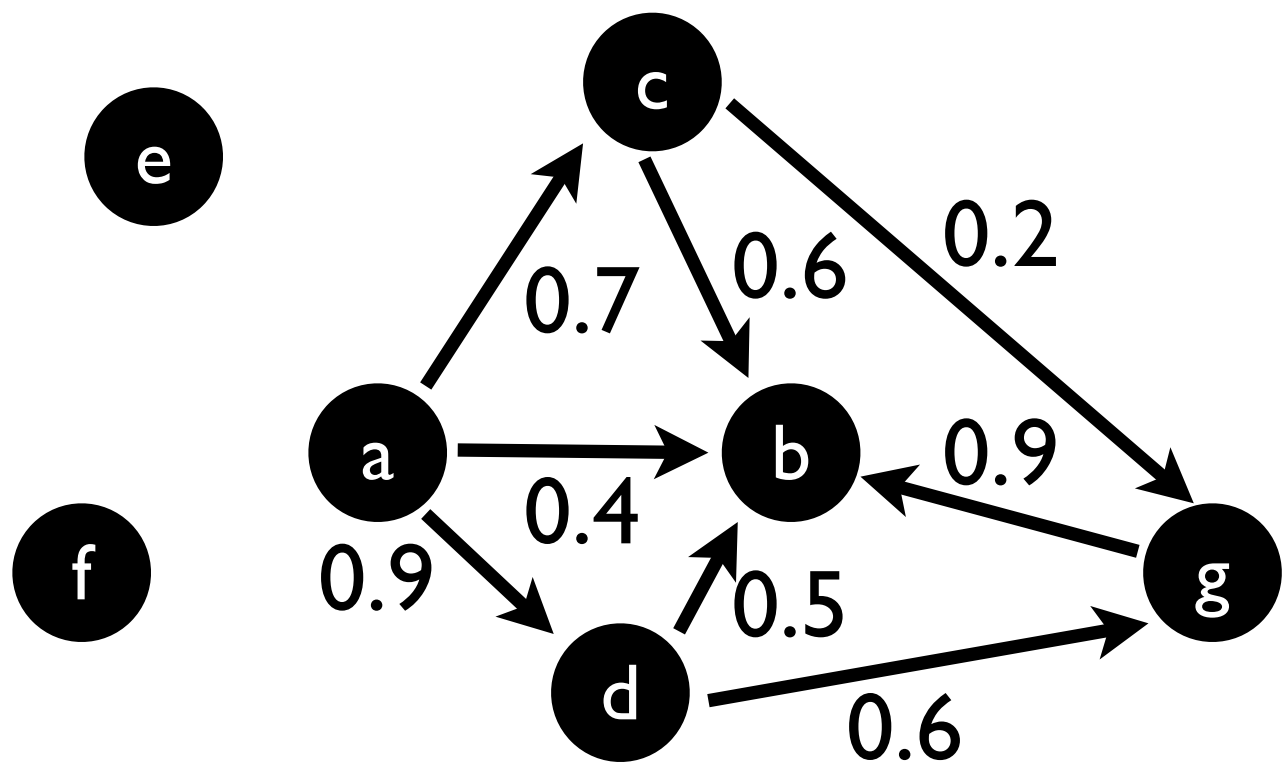
0.8 :: edge(e,c) .
0.3 :: edge(e,a) .
...

```

best subnetwork of at most
5 edges where **path(a,b)**
but not **path(e,g)**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$

Theory compression



```

path(X,Y) :- edge(X,Y) .
path(X,Y) :- edge(X,Z) , path(Z,Y) .

```

```

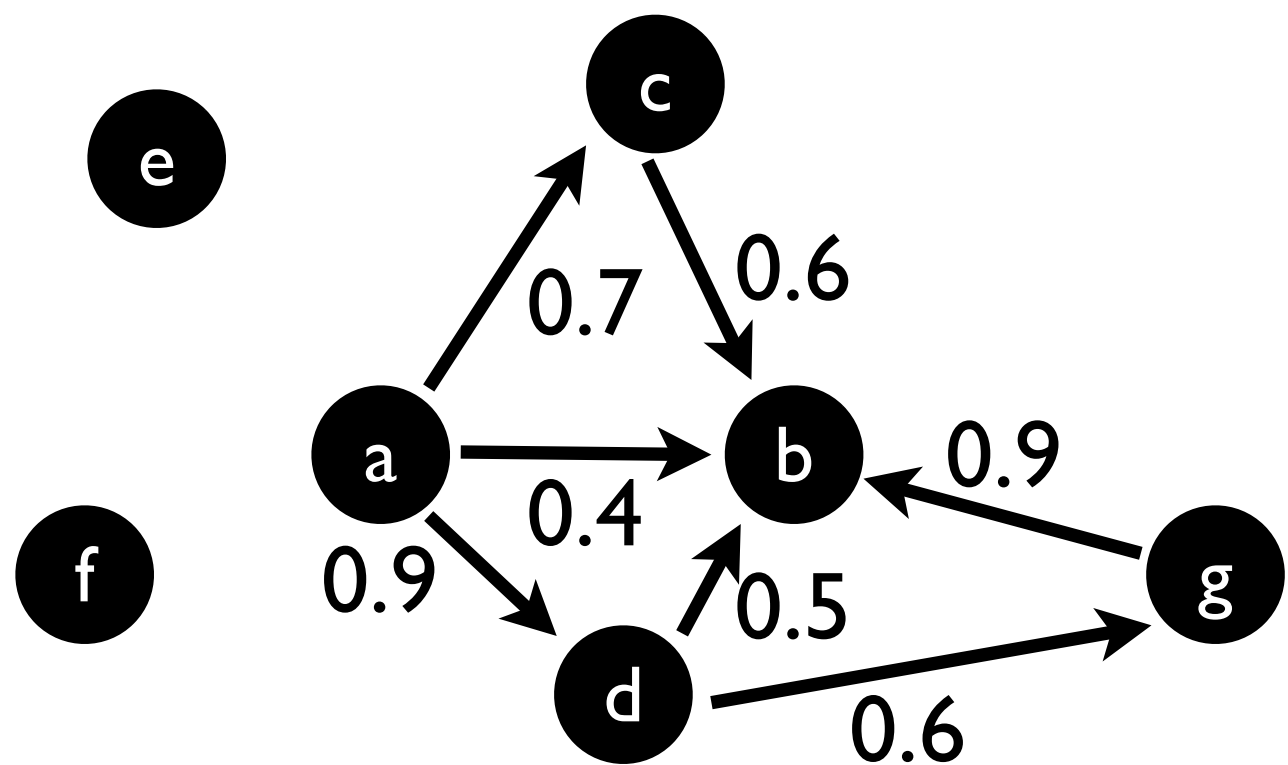
0.8 :: edge(e,c) .
0.3 :: edge(e,a) .
...

```

best subnetwork of at most
5 edges where **path(a,b)**
but not **path(e,g)**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$

Theory compression



```

path(X,Y) :- edge(X,Y) .
path(X,Y) :- edge(X,Z) , path(Z,Y) .

```

```

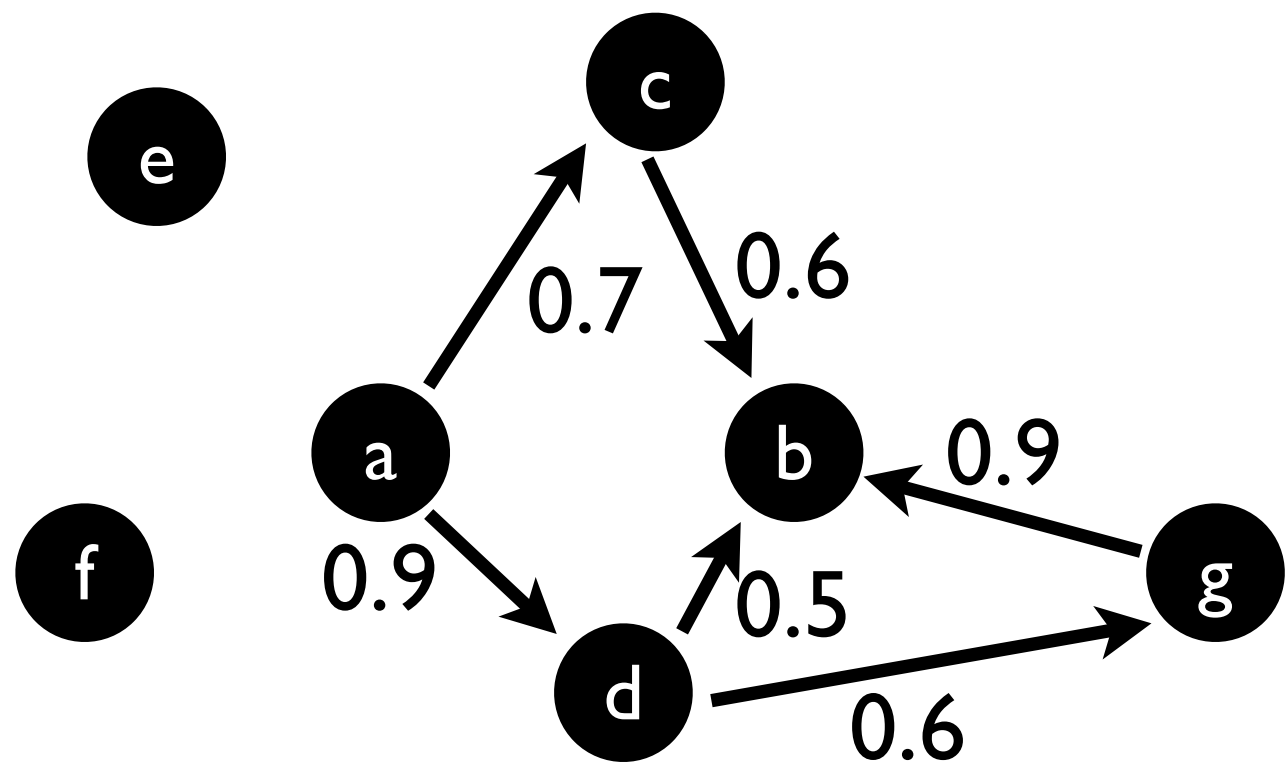
0.8 :: edge(e,c) .
0.3 :: edge(e,a) .
...

```

best subnetwork of at most
5 edges where **path(a,b)**
but not **path(e,g)**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$

Theory compression



```

path(X,Y) :- edge(X,Y) .
path(X,Y) :- edge(X,Z) , path(Z,Y) .

```

```

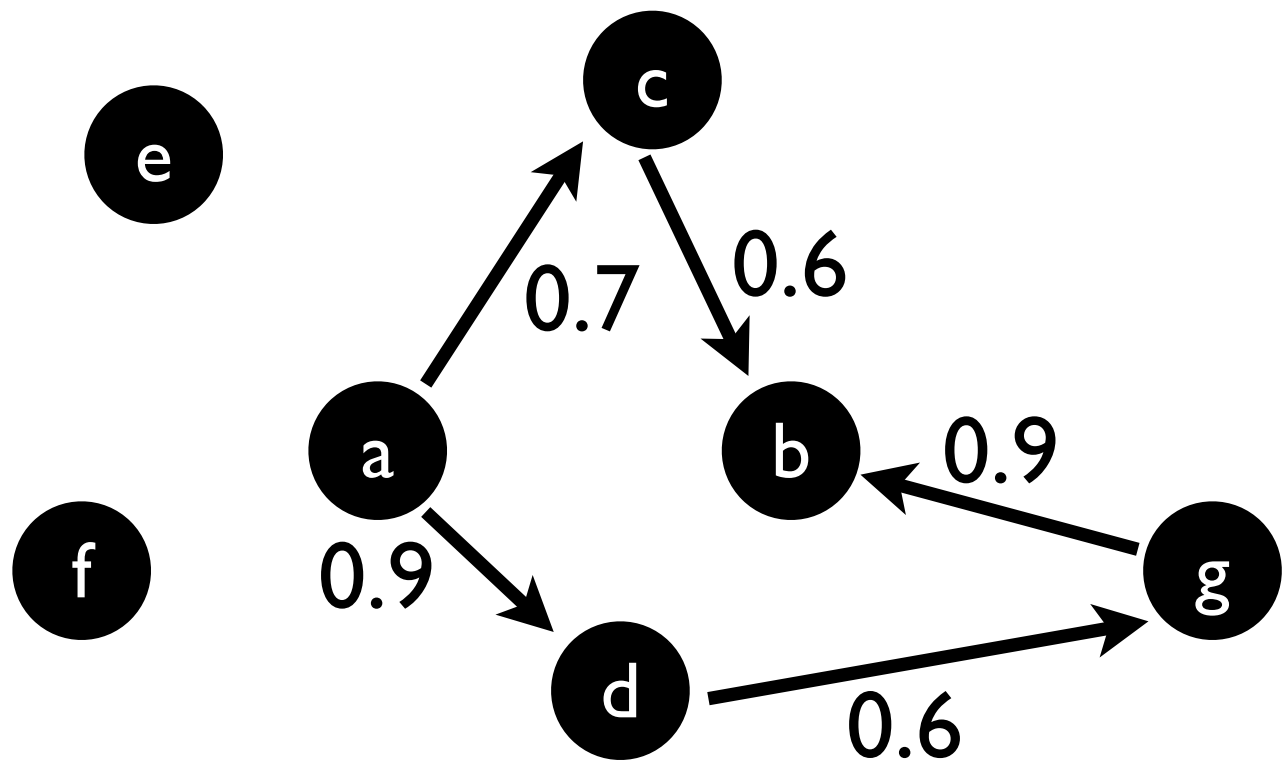
0.8 :: edge(e,c) .
0.3 :: edge(e,a) .
...

```

best subnetwork of at most
5 edges where **path(a,b)**
but not **path(e,g)**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$

Theory compression



```

path(X,Y) :- edge(X,Y) .
path(X,Y) :- edge(X,Z) , path(Z,Y) .

```

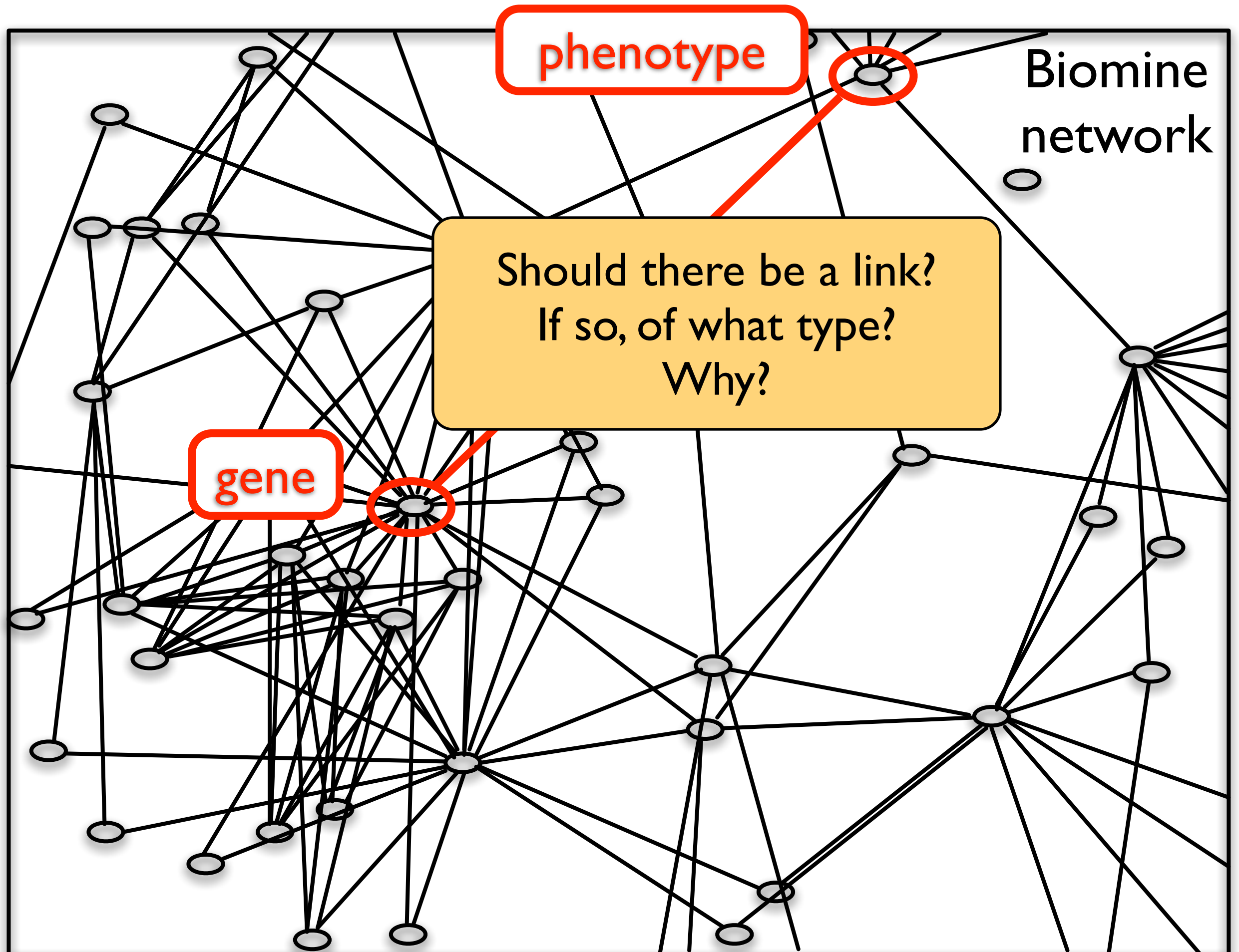
```

0.8::edge(e,c) .
0.3::edge(e,a) .
...

```

best subnetwork of at most
5 edges where **path(a,b)**
but not **path(e,g)**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$



Learning Patterns

- **Probabilistic Explanation Based Learning:**
example + background theory \rightarrow rule
- **Probabilistic Query Mining:**
pos./neg. examples \rightarrow set of independent rules
- **Probabilistic Rule Learning:**
probabilistic examples \rightarrow (probabilistic) concept definition

ProbFOIL

- Upgrading FOIL to learn a set of rules from **probabilistic** facts

ProbFOIL

- Upgrading FOIL to learn a set of rules from **probabilistic** facts

```
0.4987::rain(1) .  
0.3591::windok(1) .  
0.4534::sunshine(1) .  
0.3257::surfing(1) .  
0.7391::rain(2) .  
0.6022::windok(2) .  
0.9837::sunshine(2) .  
0.2592::surfing(2) .  
0.2898::rain(3) .  
0.7423::windok(3) .  
0.2275::sunshine(3) .  
0.5688::surfing(3) .  
...
```

ProbFOIL

- Upgrading FOIL to learn a set of rules from **probabilistic** facts

```
0.4987::rain(1) .
0.3591::windok(1) .
0.4534::sunshine(1) .
0.3257::surfing(1) .
0.7391::rain(2) .
0.6022::windok(2) .
0.9837::sunshine(2) .
0.2592::surfing(2) .
0.2898::rain(3) .
0.7423::windok(3) .
0.2275::sunshine(3) .
0.5688::surfing(3) .
...
```

```
surfing(X) :-
    \+ rain(X) , windok(X) .
surfing(X) :-
    \+ rain(X) , sunshine(X) .
```

ProbFOIL

- Upgrading FOIL to learn a set of rules from **probabilistic** facts

```
0.7::father(piet, wim) .  
0.9::father(bart, pieter) .  
0.6::father(tom, greet) .  
    mother(emma, piet) .  
    mother(emma, greet) .  
    mother(greet, pieter) .  
    grandmother(emma, ilse) .  
0.7::grandmother(emma, wim) .  
    . . . .
```

ProbFOIL

- Upgrading FOIL to learn a set of rules from **probabilistic** facts

```
0.7::father(piet, wim) .
0.9::father(bart, pieter) .
0.6::father(tom, greet) .
    mother(emma, piet) .
    mother(emma, greet) .
    mother(greet, pieter) .
grandmother(emma, ilse) .
0.7::grandmother(emma, wim) .
    . . . .
```

```
grandmother(X, Y) :-
    mother(X, Z) ,
    father(Z, Y) .
```


Prob2FOIL

- Learning **probabilistic rules** from probabilistic facts

```
0.4987::rain(1) .
0.3591::windok(1) .
0.4534::sunshine(1) .
0.3257::surfing(1) .
0.7391::rain(2) .
0.6022::windok(2) .
0.9837::sunshine(2) .
0.2592::surfing(2) .
0.2898::rain(3) .
0.7423::windok(3) .
0.2275::sunshine(3) .
0.5688::surfing(3) .
...
```

Prob2FOIL

- Learning **probabilistic rules** from probabilistic facts

```
0.4987::rain(1) .
0.3591::windok(1) .
0.4534::sunshine(1) .
0.3257::surfing(1) .
0.7391::rain(2) .
0.6022::windok(2) .
0.9837::sunshine(2) .
0.2592::surfing(2) .
0.2898::rain(3) .
0.7423::windok(3) .
0.2275::sunshine(3) .
0.5688::surfing(3) .
...
```

```
0.7023::surfing(A) <-
                                \+rain(A) .
0.01243::surfing(A) <-
                                true .
```

NELL

Table 5: Number of facts per predicate (NELL athlete dataset)

athletecoach(person,person)	18	athleteplaysforteam(person,team)	721
athleteplayssport(person,sport)	1921	teamplaysinleague(team,league)	1085
athleteplaysinleague(person,league)	872	athletealsoknownas(person,name)	17
coachesinleague(person,league)	93	coachesteam(person,team)	132
teamhomestadium(team,stadium)	198	teamplayssport(team,sport)	359
athleteplayssportsteamposition(person,position)	255	athlethomestadium(person,stadium)	187
athlete(person)	1909	attraction(stadium)	2
coach(person)	624	female(person)	2
male(person)	7	hobby(sport)	5
organization(league)	1	person(person)	2
personafrica(person)	1	personasia(person)	4
personaaustralia(person)	22	personcanada(person)	1
personeurope(person)	1	personmexico(person)	108
personus(person)	6	sport(sport)	36
sportsleague(league)	18	sportsteam(team)	1330
sportsteamposition(position)	22	stadiumoreventvenue(stadium)	171

athleteplaysforteam

athleteplaysforteam(A,B) :- coachesteam(A,B).

0.875::athleteplaysforteam(A,B) :- teamhomestadium(B,C), athletehomestadium(A,C).

0.99080::athleteplaysforteam(A,B) :- teamhomestadium(B,_), male(A), athleteplayssport(A,_).

0.75::athleteplaysforteam(A,B) :- teamhomestadium(B,_), athleteplaysinleague(A,C), teamplaysinleague(B,C), athlete(A).

0.75::athleteplaysforteam(A,B) :- teamplayssport(B,C), athleteplayssport(A,C), coach(A), teamplaysinleague(B,_).

0.97555::athleteplaysforteam(A,B) :- personus(A), teamplayssport(B,_).

0.762::athleteplaysforteam(A,B) :- teamplayssport(B,C), athleteplayssport(A,C), personmexico(A), teamplaysinleague(B,_).

0.52571::athleteplaysforteam(A,B) :- teamplayssport(B,C), athleteplayssport(A,C), athleteplaysinleague(A,_), teamplaysinleague(B,_), athlete(A), teamplayssport(B,C).

0.50546::athleteplaysforteam(A,B) :- teamplayssport(B,_), teamplaysinleague(B,C), athleteplaysinleague(A,C), athleteplayssport(A,_).

0.50::athleteplaysforteam(A,B) :- teamplayssport(B,_), teamplaysinleague(B,C), athleteplaysinleague(A,C).

0.52941::athleteplaysforteam(A,B) :- teamplayssport(B,_), teamhomestadium(B,_), coach(A), teamplaysinleague(B,_).

0.55287::athleteplaysforteam(A,B) :- teamplayssport(B,_), teamplaysinleague(B,C), athleteplaysinleague(A,C), athlete(A).

0.46875::athleteplaysforteam(A,B) :- teamplayssport(B,_), teamplaysinleague(B,_), coach(A), teamhomestadium(B,_).

Complexity of Querying

Complexity of querying

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

$$0.96 \times 0.99 = 0.95$$

$$0.9 \times 0.93 = 0.83$$

$$0.87 \times 0.93 = 0.80$$

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

Complexity of querying

ProducesProduct

HeadquarteredIn

Company	Product	P	Company	City	P
s					00
m					99
i					93
m					93
a					93
a					93
.					93

```
select distinct x.Product, x.Company,
               x.P * y.P as P
from ProducesProduct x, HeadquarteredIn y
where x.Company = y.Company
      and y.City = 'san_jose'
order by P desc
```

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

$$0.96 \times 0.99 = 0.95$$

$$0.9 \times 0.93 = 0.83$$

$$0.87 \times 0.93 = 0.80$$

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

Complexity of querying

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...

```

result(Product, Company) :-
    producesProduct(Company, Product),
    headquarteredIn(Company, san_jose).
query(result(_, _)).
    
```


Complexity of querying

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...

`result(Product, Company) :-`

`producesProduct(Company, Product),`

`headquarteredIn(Company, san_jose).`

`query(result(_, _)).`

each ground query has a **single proof**

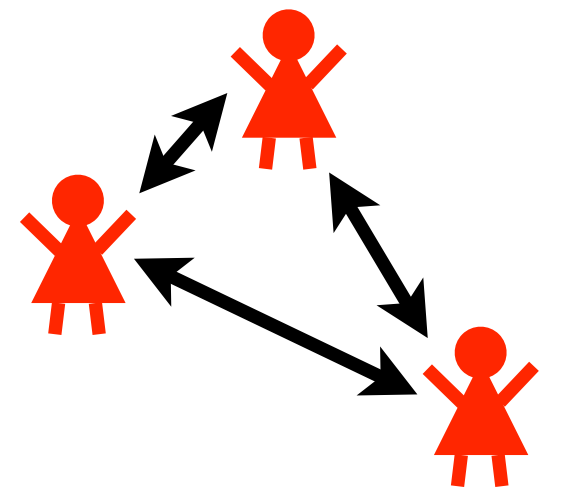
→ no disjoint-sum-problem,

easy evaluation

Complexity of querying in probabilistic databases

- queries have fixed size (no recursion)
- size of query \ll size of database
- complexity of evaluating given query measured in size of database (data complexity)
- previous example: polynomial (as all standard relational database queries)

How hard is evaluating q?



```
q :- stress(X) , influences(X,Y) .
```

```
0.7::stress(1) .
```

```
0.4::stress(2) .
```

```
0.9::stress(3) .
```

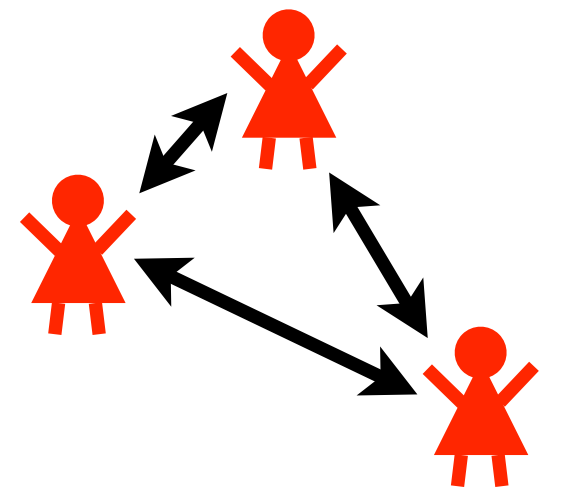
```
0.83::influences(1,2) .
```

```
0.41::influences(1,3) .
```

```
...
```

```
0.17::influences(3,2) .
```

How hard is evaluating q?



```
q :- stress(X) , influences(X,Y) .
```

```
0.7::stress(1) .
```

```
0.4::stress(2) .
```

```
0.9::stress(3) .
```

```
0.83::influences(1,2) .
```

```
0.41::influences(1,3) .
```

```
...
```

```
0.17::influences(3,2) .
```

proofs

```
s(1) , i(1,2)
```

```
s(1) , i(1,3)
```

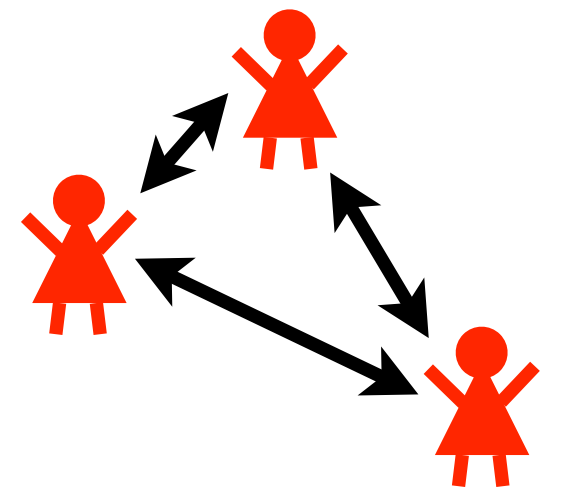
```
s(2) , i(2,1)
```

```
s(2) , i(2,3)
```

```
s(3) , i(3,1)
```

```
s(3) , i(3,2)
```

How hard is evaluating q?



`q :- stress(X) , influences(X,Y) .`

`0.7::stress(1) .`

`0.4::stress(2) .`

`0.9::stress(3) .`

`0.83::influences(1,2) .`

`0.41::influences(1,3) .`

`...`

`0.17::influences(3,2) .`

proofs

`s(1), i(1,2)`

`s(1), i(1,3)`

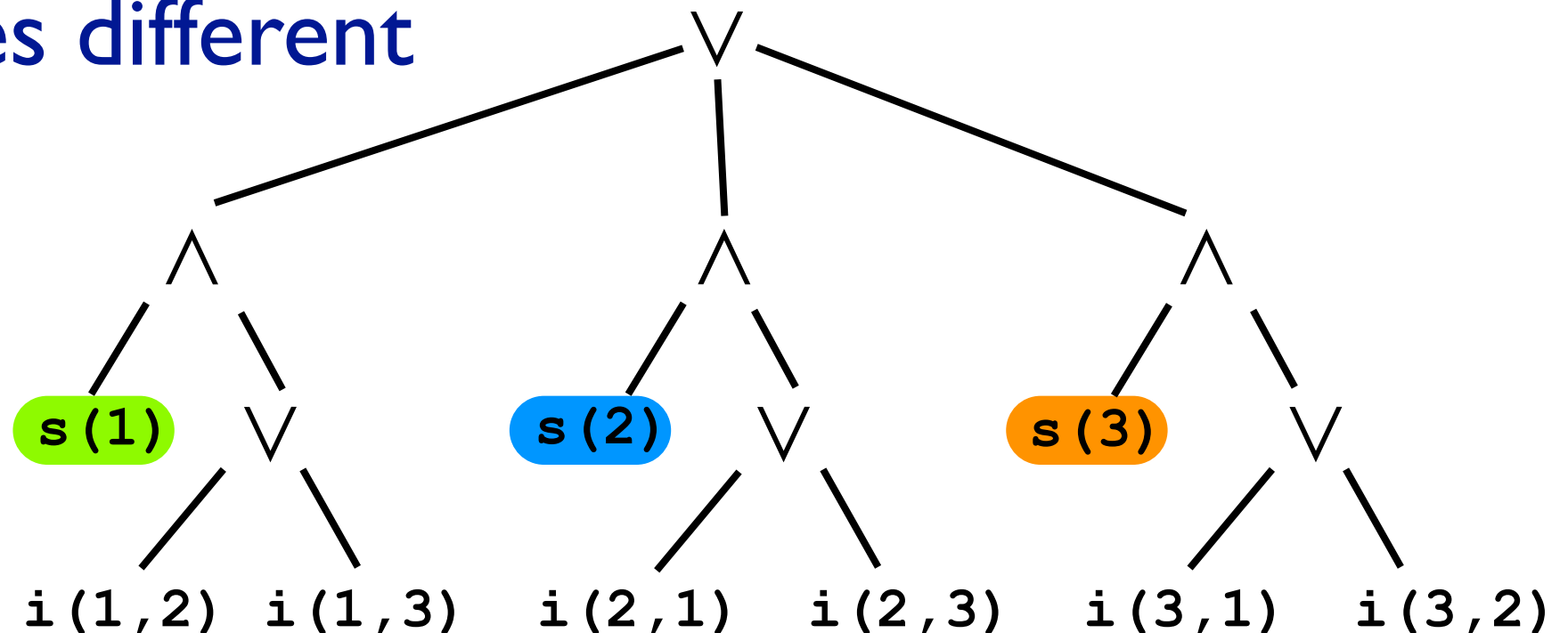
`s(2), i(2,1)`

`s(2), i(2,3)`

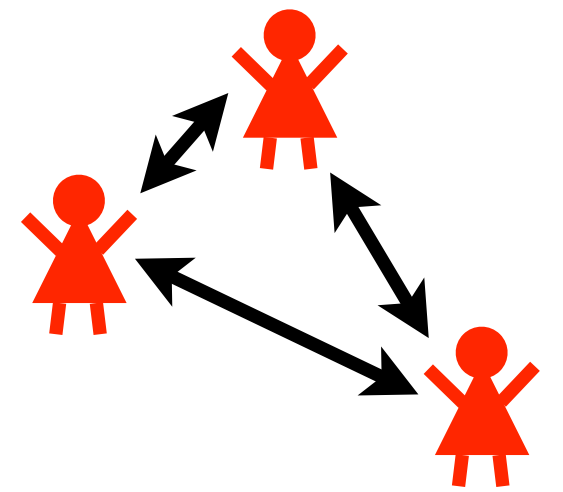
`s(3), i(3,1)`

`s(3), i(3,2)`

tree structure, all
leaves different



How hard is evaluating q?



`q :- stress(X) , influences(X,Y) .`

`0.7::stress(1) .`

`0.4::stress(2) .`

`0.9::stress(3) .`

`0.83::influences(1,2) .`

`0.41::influences(1,3) .`

`...`

`0.17::influences(3,2) .`

proofs

`s(1), i(1,2)`

`s(1), i(1,3)`

`s(2), i(2,1)`

`s(2), i(2,3)`

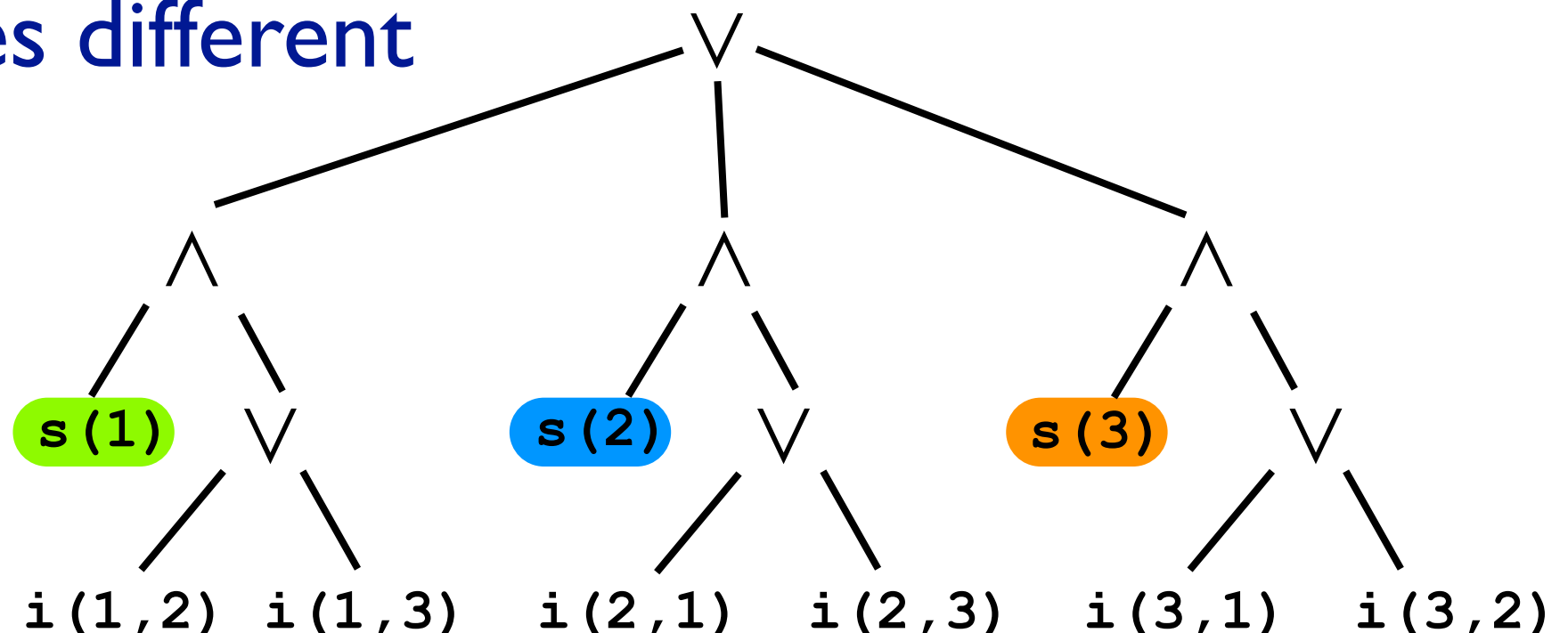
`s(3), i(3,1)`

`s(3), i(3,2)`

tree structure, all
leaves different

$$P(\varphi_1 \wedge \varphi_2) = P(\varphi_1) \cdot P(\varphi_2)$$

$$P(\varphi_1 \vee \varphi_2) = 1 - (1 - P(\varphi_1)) \cdot (1 - P(\varphi_2))$$



$$P(q) = 1 - \prod_{j=1..n} \left(1 - \left(P(s(X_j)) \left(1 - \prod_{k=1..n, k \neq j} (1 - P(i(X_j, Y_k))) \right) \right) \right)$$

polynomial in database size / number of persons n

proofs

$s(1), i(1, 2)$

$s(1), i(1, 3)$

$s(2), i(2, 1)$

$s(2), i(2, 3)$

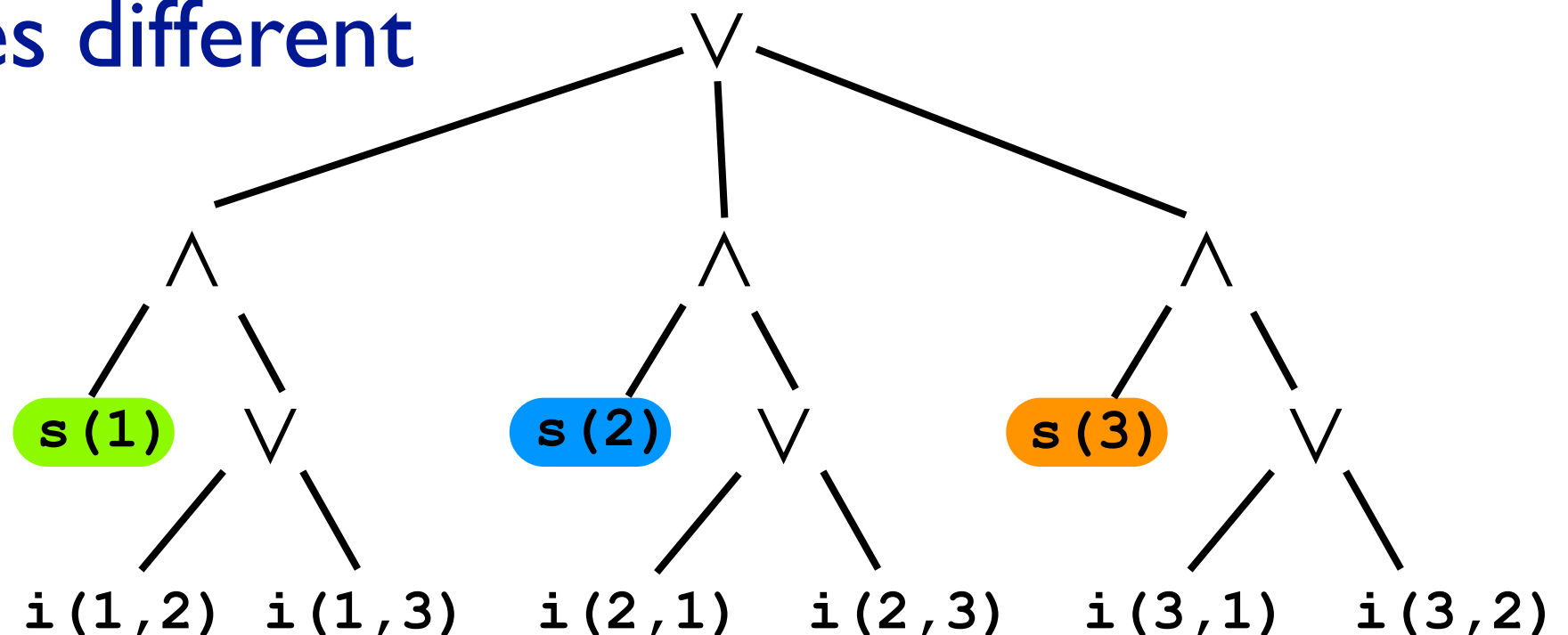
$s(3), i(3, 1)$

$s(3), i(3, 2)$

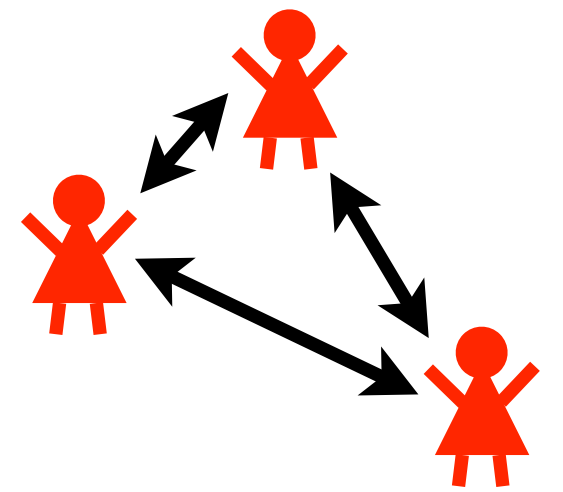
tree structure, all
leaves different

$$P(\varphi_1 \wedge \varphi_2) = P(\varphi_1) \cdot P(\varphi_2)$$

$$P(\varphi_1 \vee \varphi_2) = 1 - (1 - P(\varphi_1)) \cdot (1 - P(\varphi_2))$$



How hard is evaluating q?



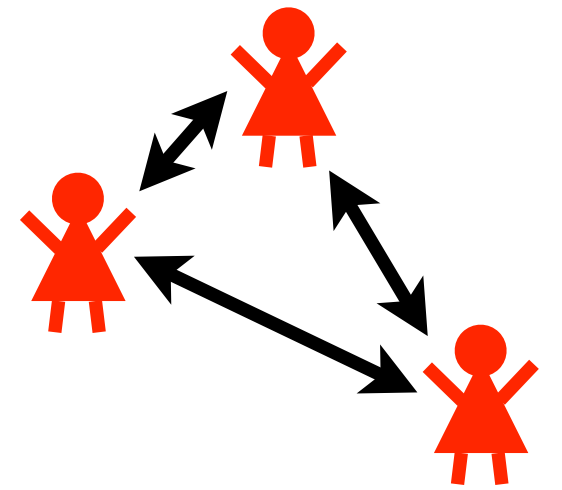
```
q :- stress(X) , influences(X,Y) , male(Y) .
```

```
0.3::male(1) .  
0.8::male(2) .  
0.9::male(3) .
```

```
0.7::stress(1) .  
0.4::stress(2) .  
0.9::stress(3) .
```

```
0.83::influences(1,2) .  
0.41::influences(1,3) .  
...  
0.17::influences(3,2) .
```

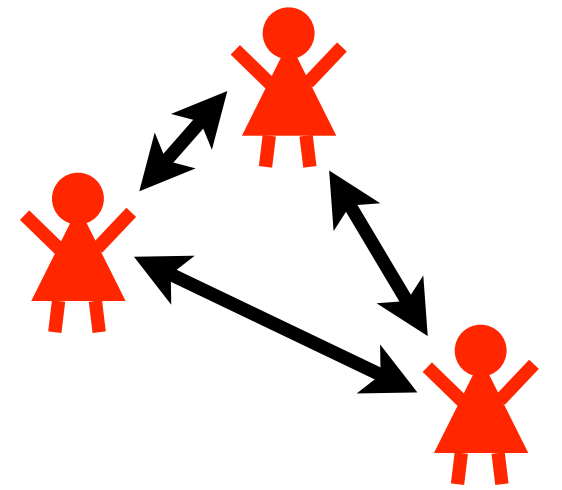

How hard is evaluating q?



```
q :- stress(X) , influences(X,Y) , male(Y) .
```

0.3::male(1) .	0.7::stress(1) .	0.83::influences(1,2) .
0.8::male(2) .	0.4::stress(2) .	0.41::influences(1,3) .
0.9::male(3) .	0.9::stress(3)
		0.17::influences(3,2) .

How hard is evaluating q?



```
q :- stress(X) , influences(X,Y) , male(Y) .
```

0.3::male(1) .	0.7::stress(1) .	0.83::influences(1,2) .
0.8::male(2) .	0.4::stress(2) .	0.41::influences(1,3) .
0.9::male(3) .	0.9::stress(3)
		0.17::influences(3,2) .

proofs

```
s(1) , i(1,2) , m(2)
```

```
s(1) , i(1,3) , m(3)
```

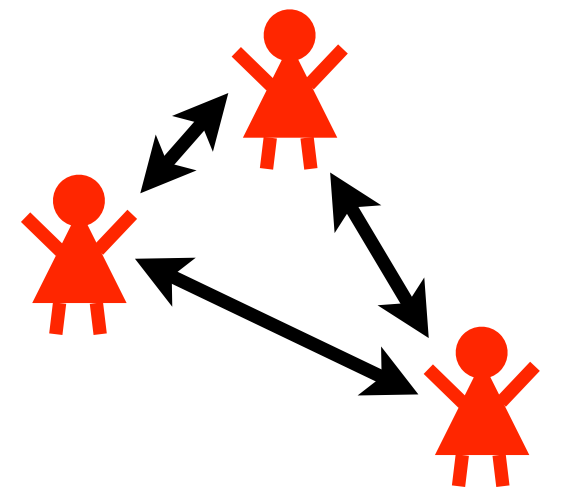
```
s(2) , i(2,1) , m(1)
```

```
s(2) , i(2,3) , m(3)
```

```
s(3) , i(3,1) , m(1)
```

```
s(3) , i(3,2) , m(2)
```

How hard is evaluating q?



```
q :- stress(X) , influences(X,Y) , male(Y) .
```

0.3::male(1) .	0.7::stress(1) .	0.83::influences(1,2) .
0.8::male(2) .	0.4::stress(2) .	0.41::influences(1,3) .
0.9::male(3) .	0.9::stress(3)
		0.17::influences(3,2) .

proofs

s(1), i(1,2), m(2)

s(1), i(1,3), m(3)

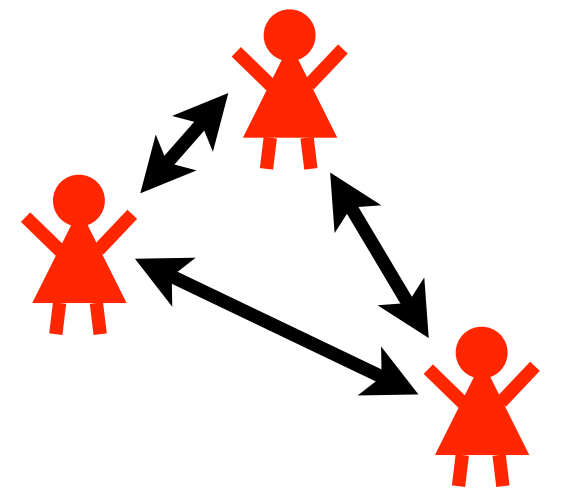
s(2), i(2,1), m(1)

s(2), i(2,3), m(3)

s(3), i(3,1), m(1)

s(3), i(3,2), m(2)

How hard is evaluating q?

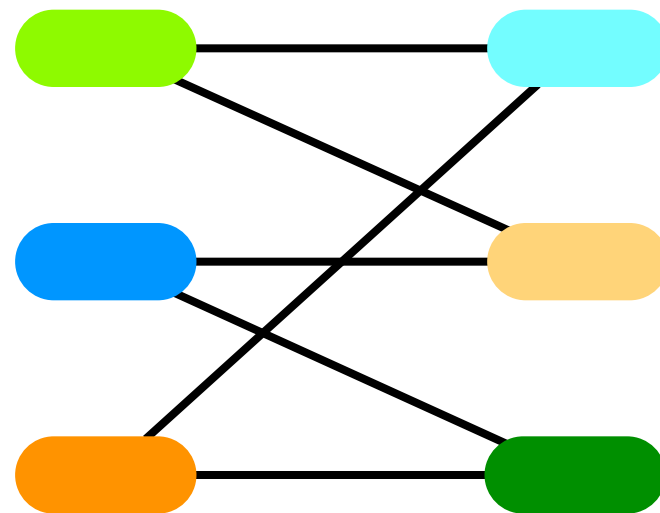


```
q :- stress(X), influences(X,Y), male(Y).
```

		0.83::influences(1,2).
0.3::male(1).	0.7::stress(1).	0.41::influences(1,3).
0.8::male(2).	0.4::stress(2).	...
0.9::male(3).	0.9::stress(3).	0.17::influences(3,2).

proofs

s(1)	, i(1,2)	, m(2)
s(1)	, i(1,3)	, m(3)
s(2)	, i(2,1)	, m(1)
s(2)	, i(2,3)	, m(3)
s(3)	, i(3,1)	, m(1)
s(3)	, i(3,2)	, m(2)



cannot build tree
structure with all
leaves different

Read-Once Formulas

- Propositional formulas that can be rewritten such that each variable occurs at most once (= as tree with all leaves different)

Read-Once Formulas

- Propositional formulas that can be rewritten such that each variable occurs at most once (= as tree with all leaves different)
- Can be evaluated in polynomial time


Read-Once Formulas

- Propositional formulas that can be rewritten such that each variable occurs at most once (= as tree with all leaves different)
- Can be evaluated in polynomial time
- Unate formula: read once \leftrightarrow P4-free & normal

Read-Once Formulas

- Propositional formulas that can be rewritten such that each variable occurs at most once (= as tree with all leaves different)
- Can be evaluated in polynomial time
- Unate formula: read once \leftrightarrow P4-free & normal


no variable
appears both
pos & neg



Read-Once Formulas

- Propositional formulas that can be rewritten such that each variable occurs at most once (= as tree with all leaves different)
- Can be evaluated in polynomial time
- Unate formula: read once \leftrightarrow P4-free & normal

no variable
appears both
pos & neg



$X \vee Y$ is unate
 $(X \vee Y) \wedge (\neg X \vee Z)$ not

P4-free and normal

- represent DNF formula as graph:
 - a node for each variable
 - edge $(X,Y) \leftrightarrow X,Y$ in same conjunct

P4-free and normal

- represent DNF formula as graph:
 - a node for each variable
 - edge $(X,Y) \leftrightarrow X,Y$ in same conjunct
- **normal**: each clique is part of a conjunct

P4-free and normal

- represent DNF formula as graph:
 - a node for each variable
 - edge $(X,Y) \leftrightarrow X,Y$ in same conjunct
- **normal**: each clique is part of a conjunct
- **P4-free**: no induced 4 node subgraph is a path

Our first example

$s(1), i(1,2)$

$s(1), i(1,3)$

$s(2), i(2,1)$

$s(2), i(2,3)$

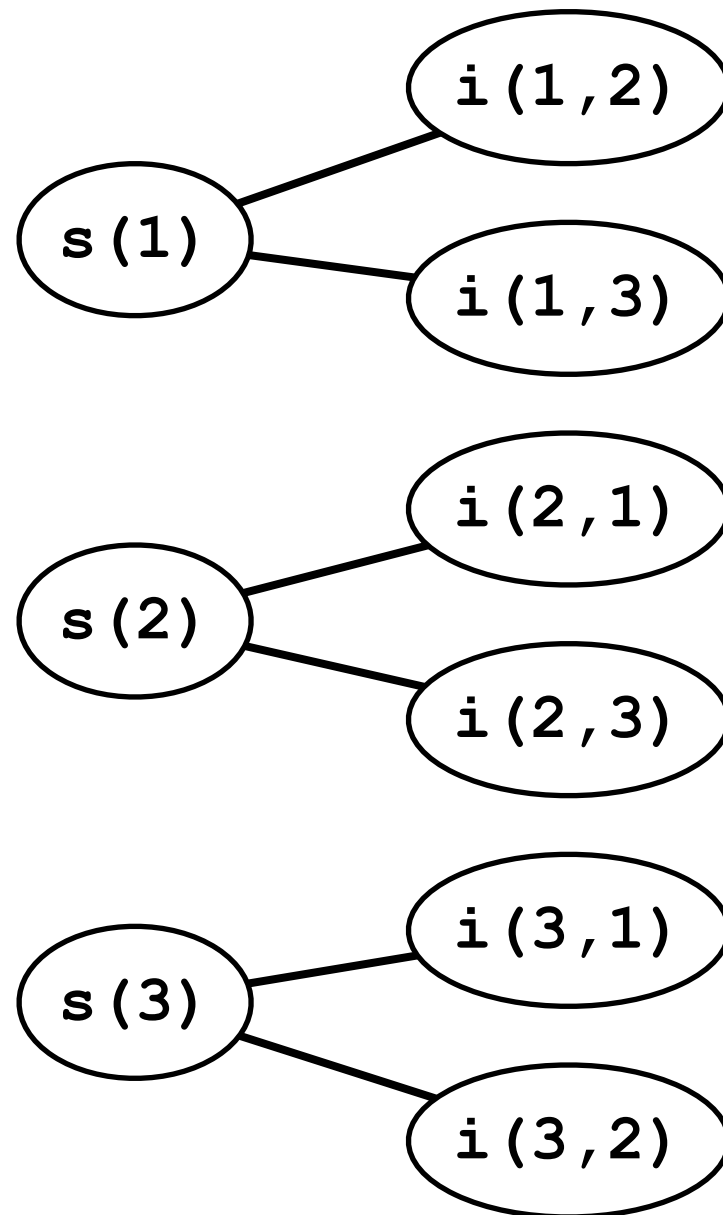
$s(3), i(3,1)$

$s(3), i(3,2)$

- **normal**: each clique is part of a conjunct
- **P4-free**: no induced 4 node subgraph is a path

Our first example

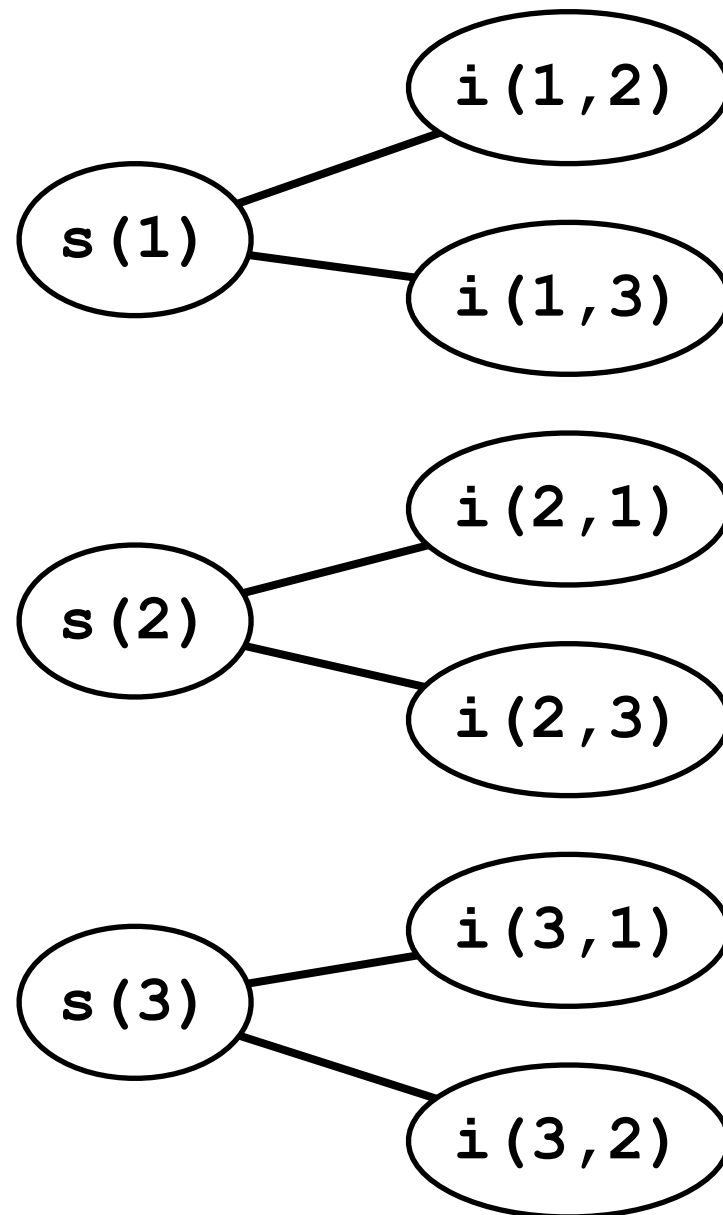
$s(1), i(1,2)$
 $s(1), i(1,3)$
 $s(2), i(2,1)$
 $s(2), i(2,3)$
 $s(3), i(3,1)$
 $s(3), i(3,2)$



- **normal**: each clique is part of a conjunct
- **P4-free**: no induced 4 node subgraph is a path

Our first example

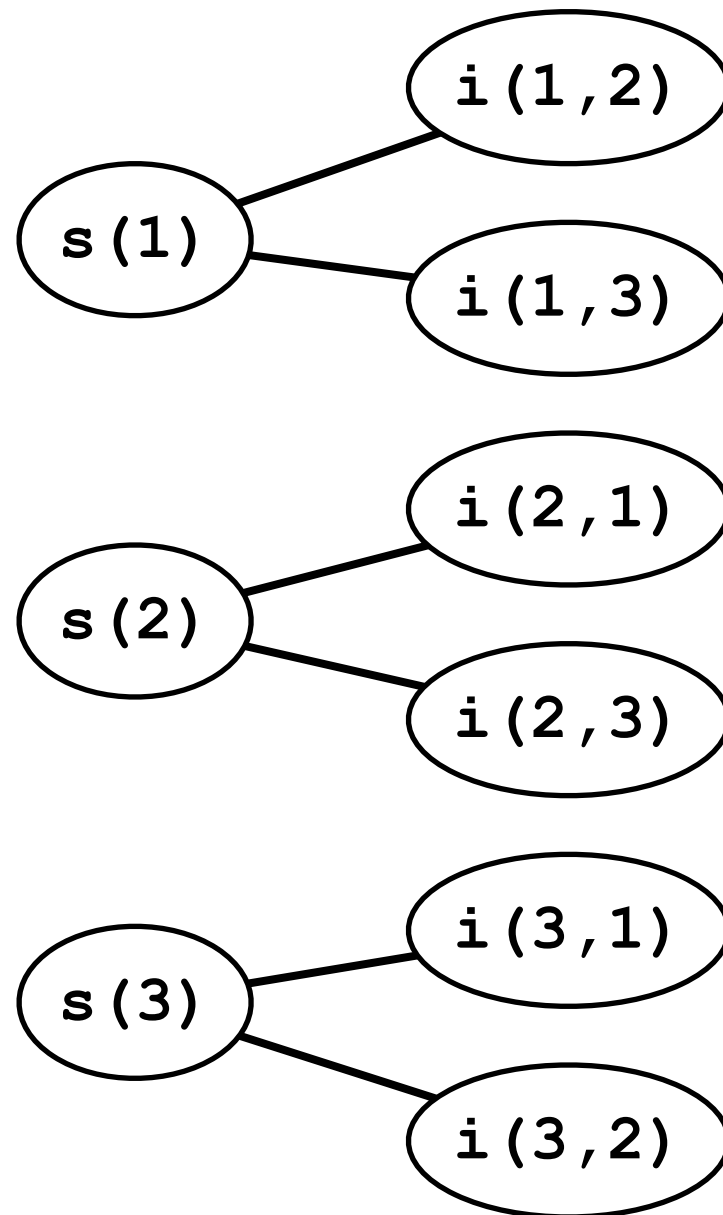
$s(1), i(1,2)$
 $s(1), i(1,3)$
 $s(2), i(2,1)$
 $s(2), i(2,3)$
 $s(3), i(3,1)$
 $s(3), i(3,2)$



- **normal**: each clique is part of a conjunct ✓
- **P4-free**: no induced 4 node subgraph is a path

Our first example

$s(1), i(1,2)$
 $s(1), i(1,3)$
 $s(2), i(2,1)$
 $s(2), i(2,3)$
 $s(3), i(3,1)$
 $s(3), i(3,2)$



- **normal**: each clique is part of a conjunct ✓
- **P4-free**: no induced 4 node subgraph is a path ✓

read-once

Our second example

$s(1), i(1,2), m(2)$

$s(1), i(1,3), m(3)$

$s(2), i(2,1), m(1)$

$s(2), i(2,3), m(3)$

$s(3), i(3,1), m(1)$

$s(3), i(3,2), m(2)$

- **normal**: each clique is part of a conjunct
- **P4-free**: no induced 4 node subgraph is a path

Our second example

$s(1), i(1,2), m(2)$

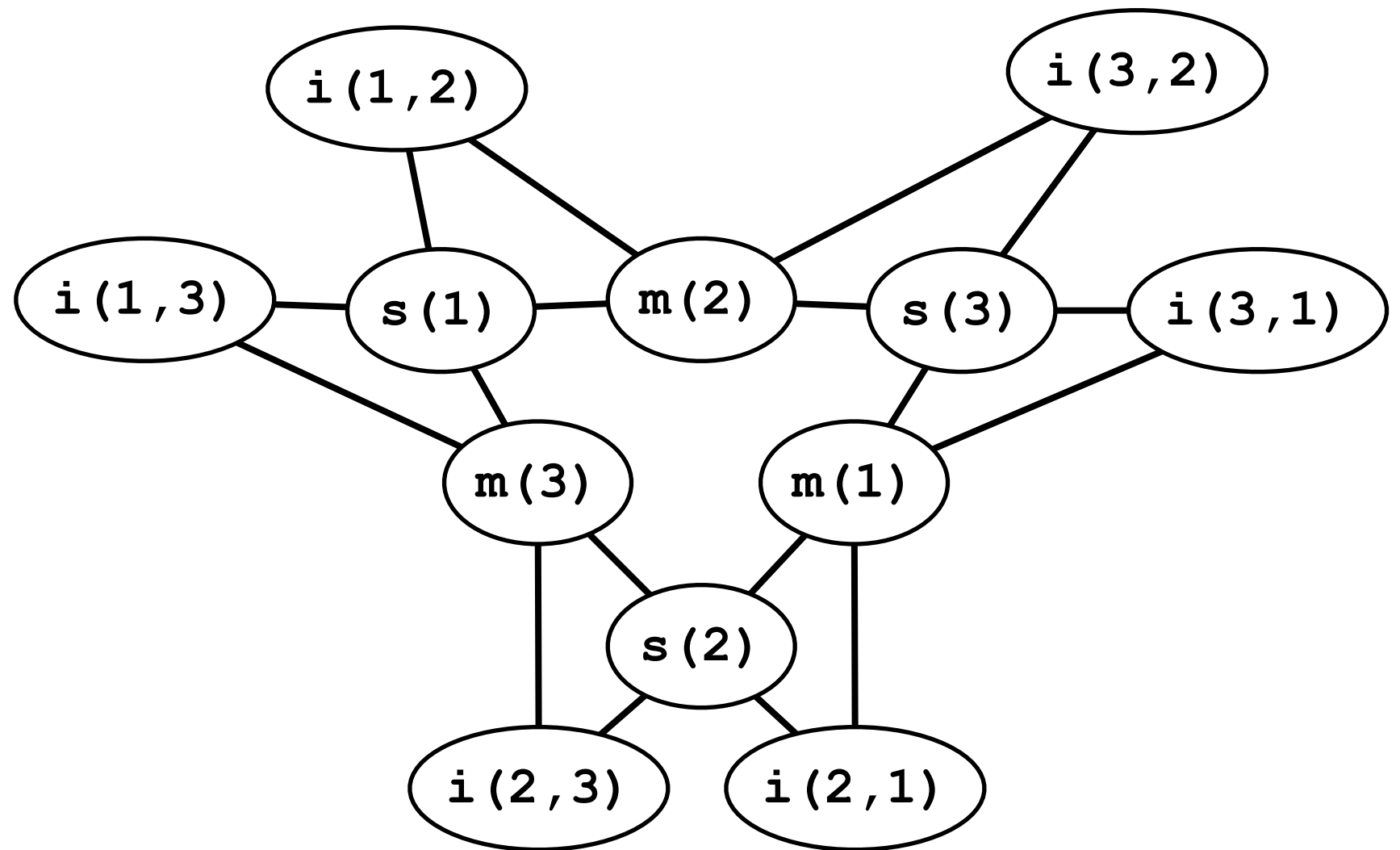
$s(1), i(1,3), m(3)$

$s(2), i(2,1), m(1)$

$s(2), i(2,3), m(3)$

$s(3), i(3,1), m(1)$

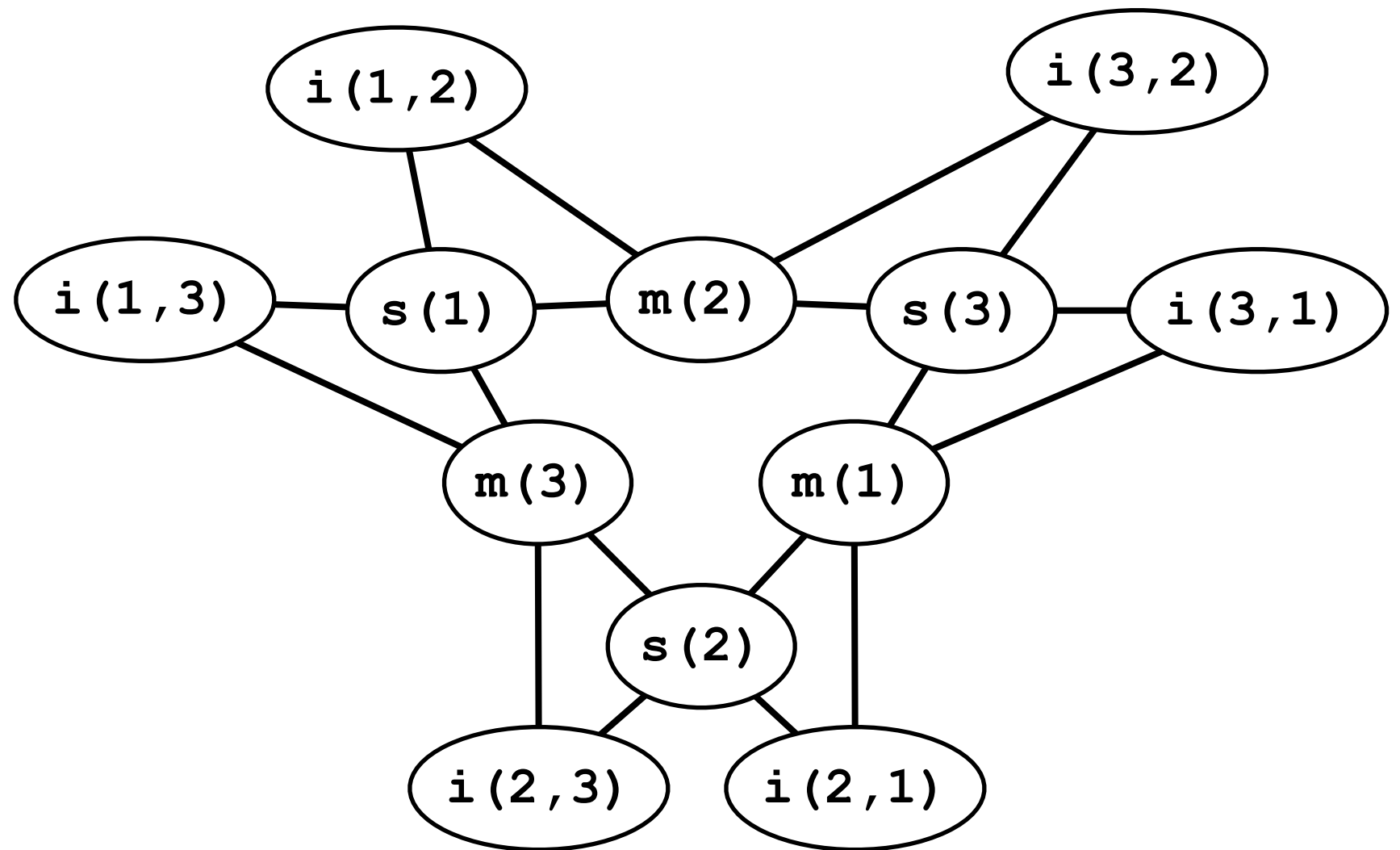
$s(3), i(3,2), m(2)$



- **normal**: each clique is part of a conjunct
- **P4-free**: no induced 4 node subgraph is a path

Our second example

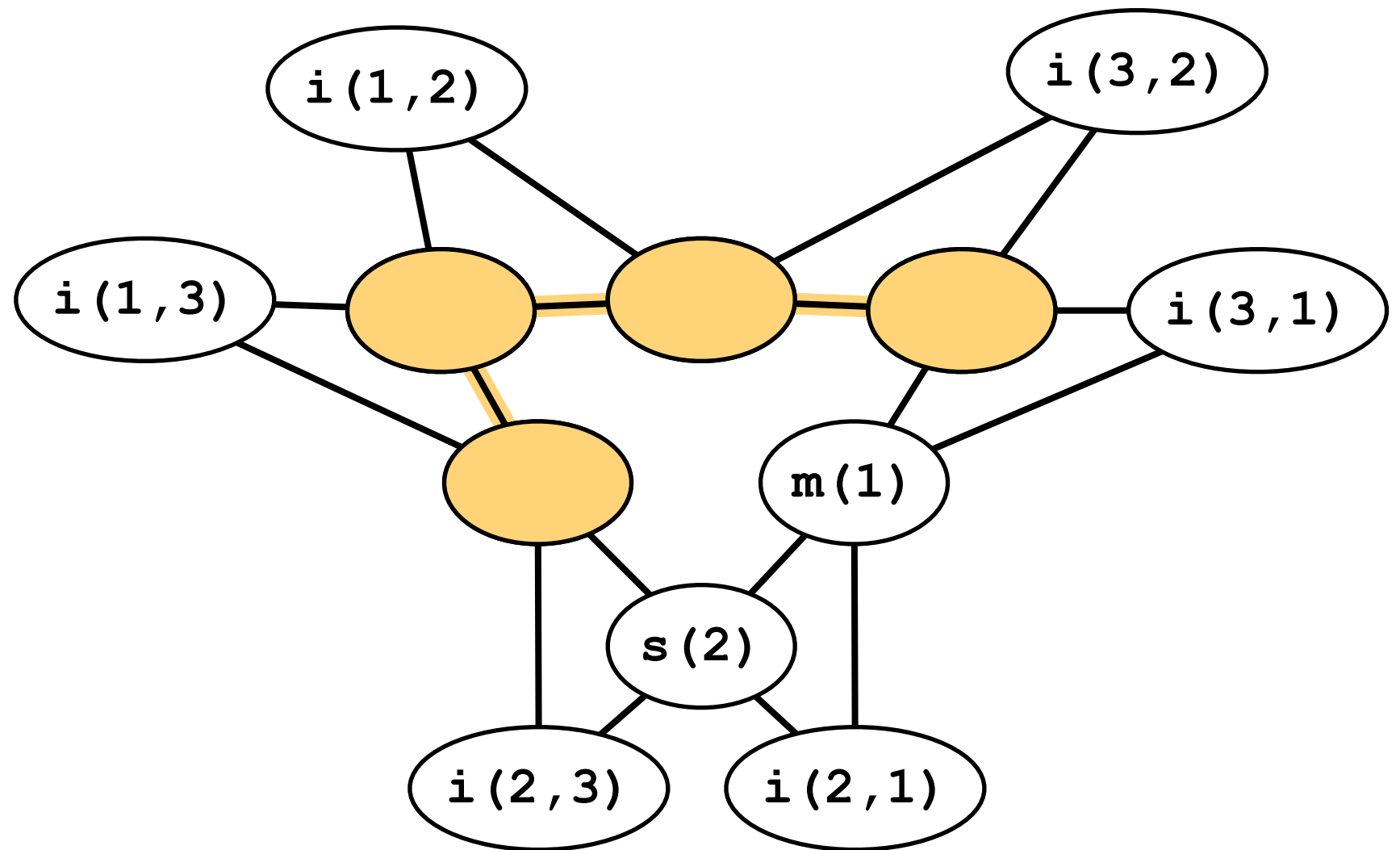
$s(1), i(1,2), m(2)$
 $s(1), i(1,3), m(3)$
 $s(2), i(2,1), m(1)$
 $s(2), i(2,3), m(3)$
 $s(3), i(3,1), m(1)$
 $s(3), i(3,2), m(2)$



- **normal**: each clique is part of a conjunct ✓
- **P4-free**: no induced 4 node subgraph is a path

Our second example

$s(1), i(1,2), m(2)$
 $s(1), i(1,3), m(3)$
 $s(2), i(2,1), m(1)$
 $s(2), i(2,3), m(3)$
 $s(3), i(3,1), m(1)$
 $s(3), i(3,2), m(2)$



- **normal**: each clique is part of a conjunct ✓
- **P4-free**: no induced 4 node subgraph is a path ✗

not read-once
(and in fact known to be
#P-hard, cf. PDB-book)

Dichotomy of UCQ Evaluation

- **U**nion of **C**onjunctive **Q**ueries
 \approx Datalog without recursion and negation
- Theorem: UCQ evaluation is either polynomial in database size or #P-hard

Dichotomy of UCQ Evaluation

- **U**nion of **C**onjunctive **Q**ueries
 \approx Datalog without recursion and negation
- Theorem: UCQ evaluation is either polynomial in database size or #P-hard



counting version of NP decision problems, e.g., model counting

#P-hard

polynomial

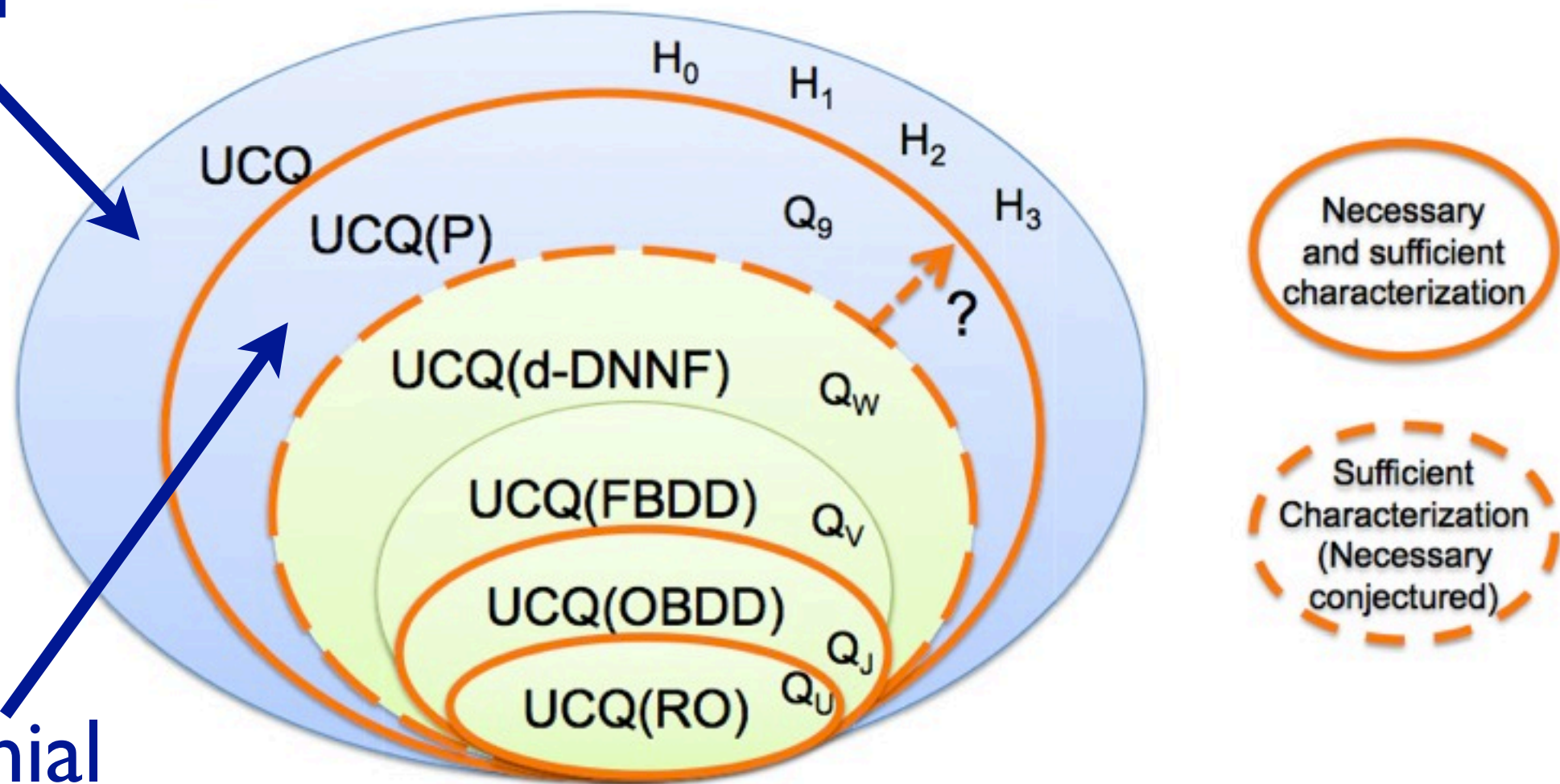


Figure 5.4: The query compilation hierarchy for Unions of Conjunctive Queries (UCQ).

Fig. from [Suciu et al 2011]

#P-hard

polynomial

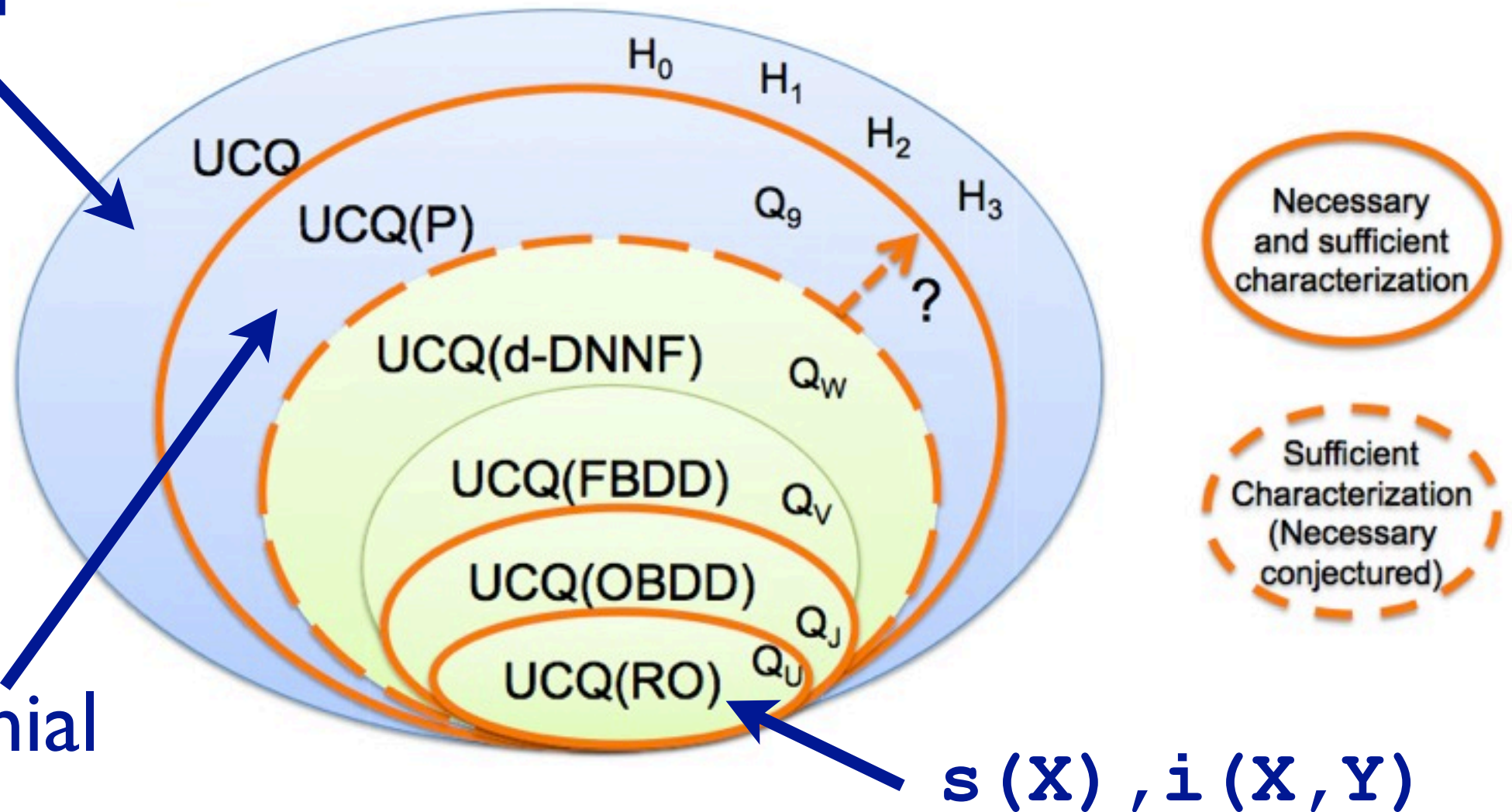


Figure 5.4: The query compilation hierarchy for Unions of Conjunctive Queries (UCQ).

Fig. from [Suciu et al 2011]

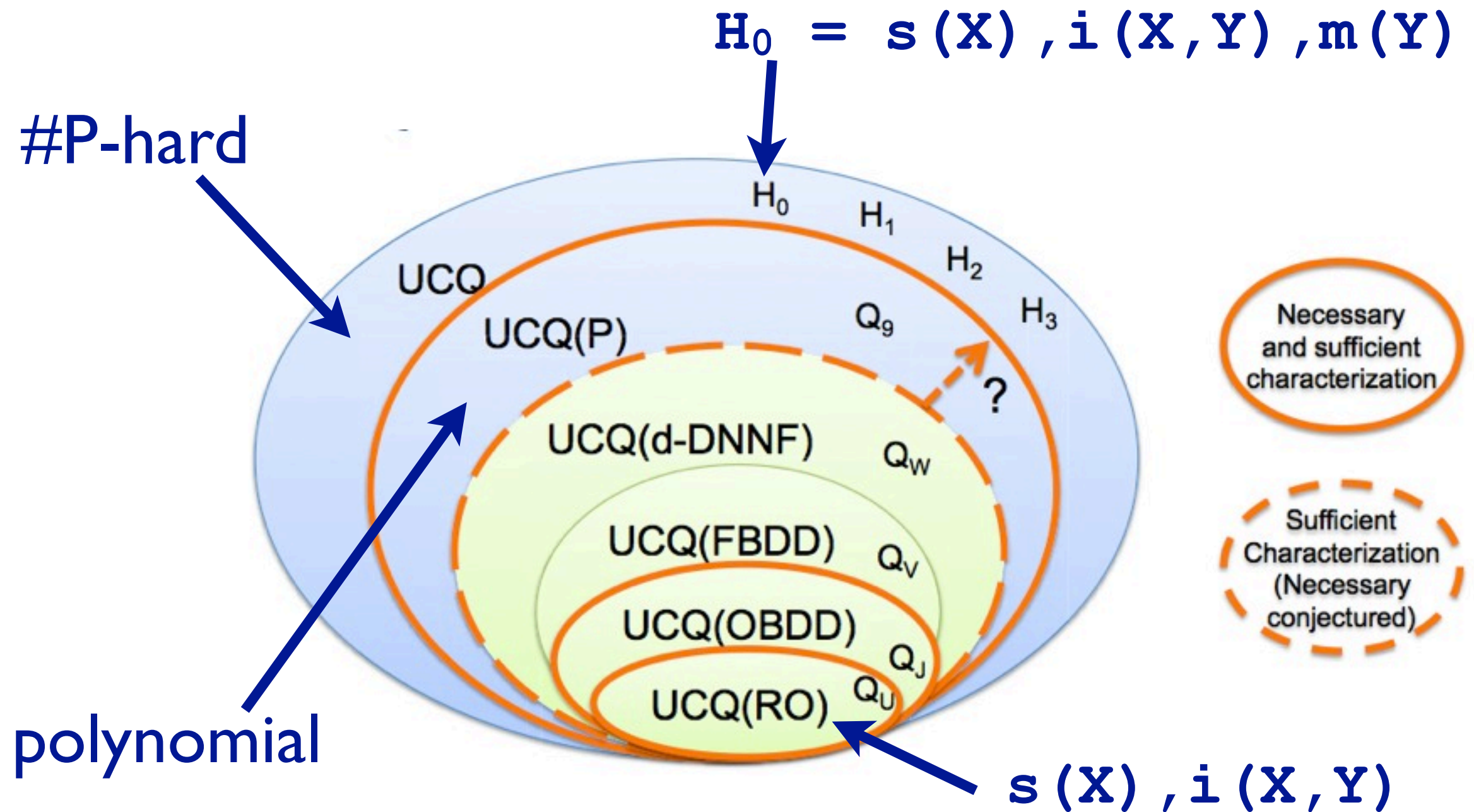


Figure 5.4: The query compilation hierarchy for Unions of Conjunctive Queries (UCQ).

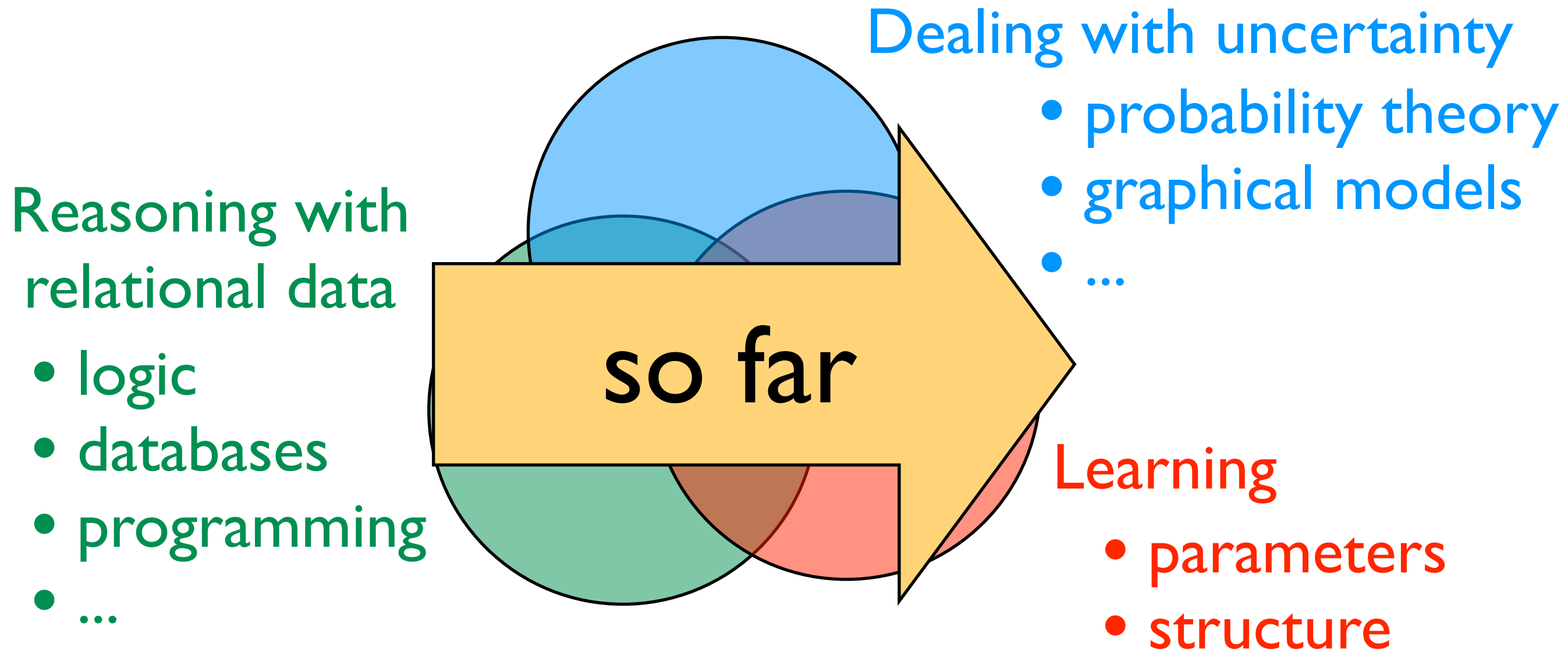
Fig. from [Suciu et al 2011]

A key question in AI:



Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

A key question in AI:



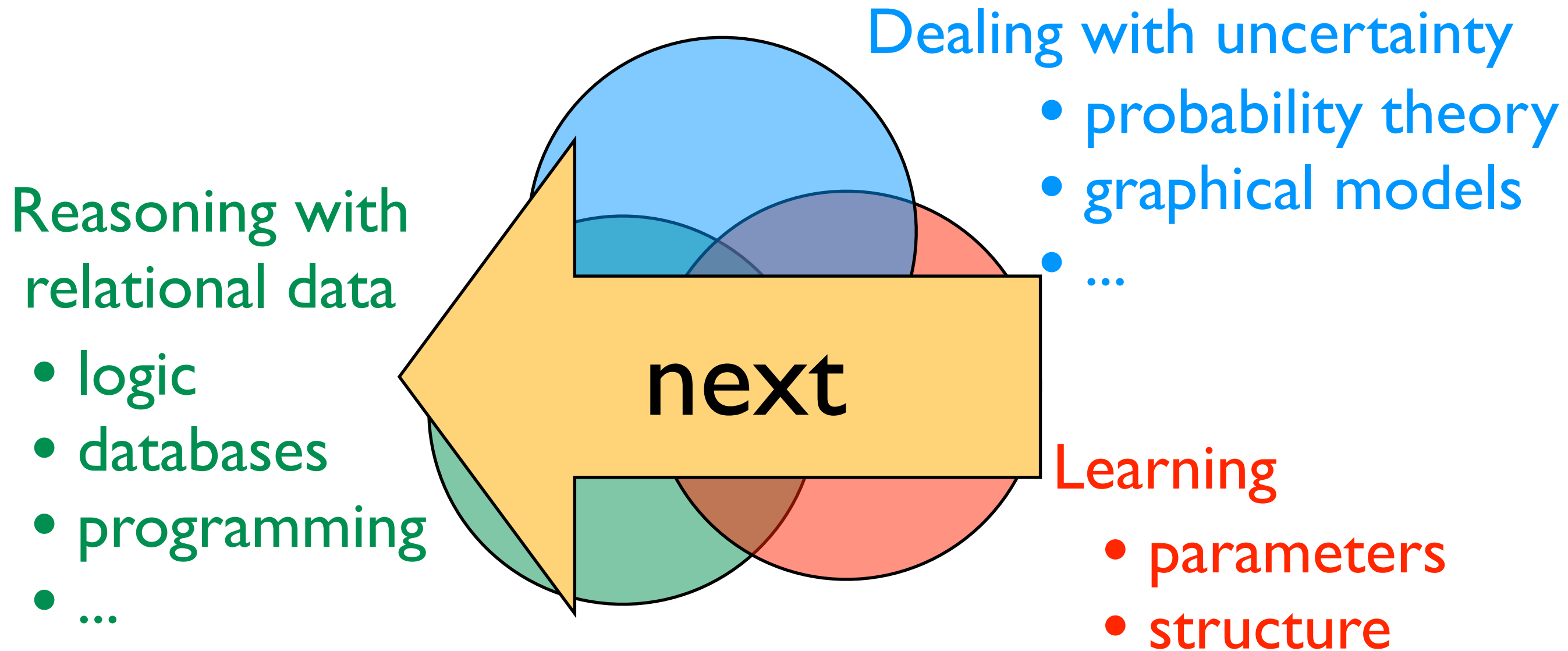
Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

A key question in AI:



Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

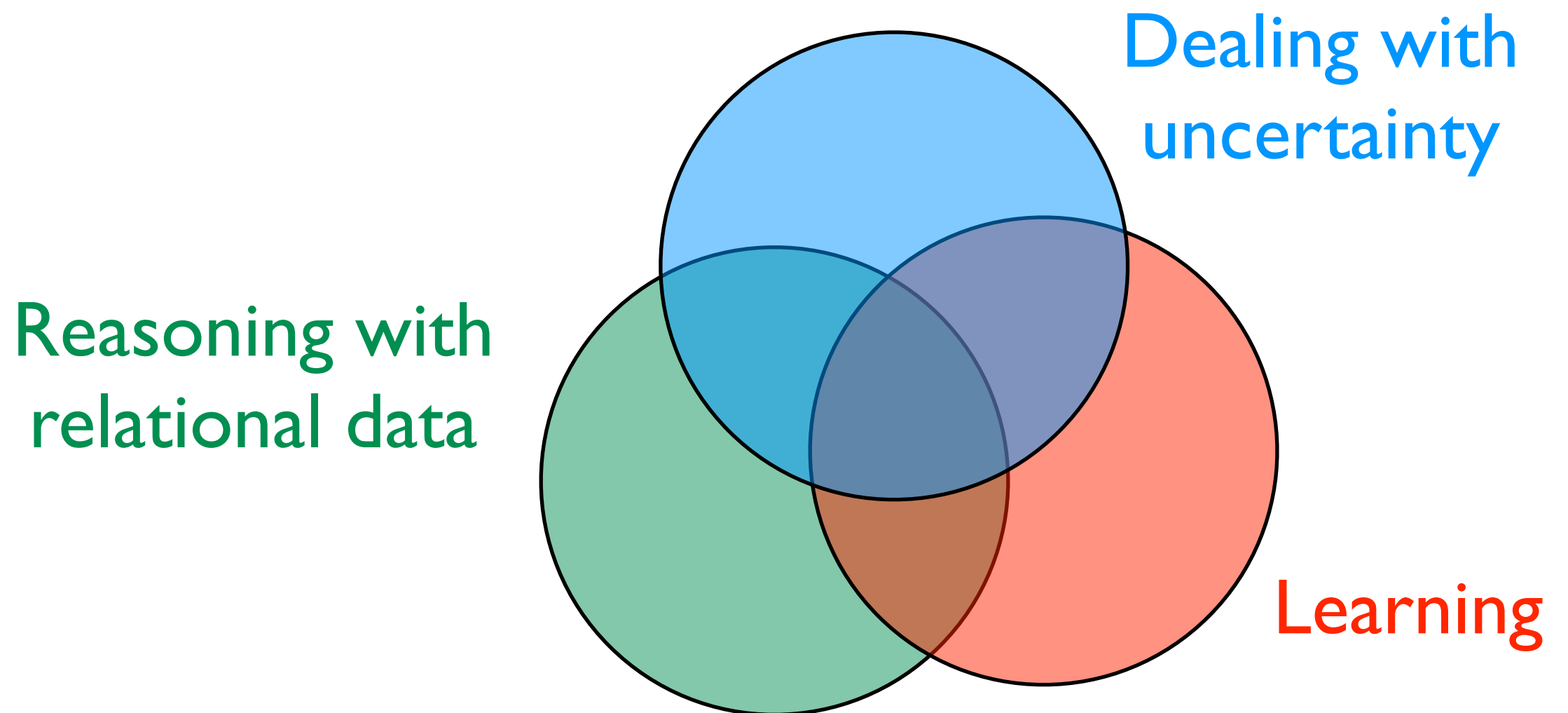
A key question in AI:



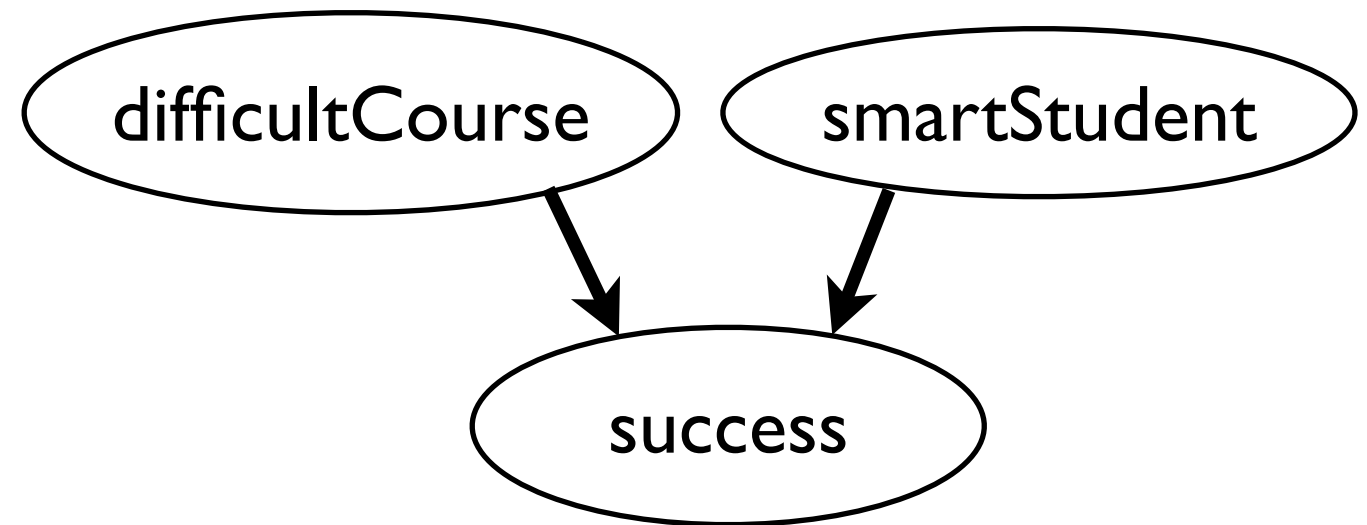
Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

(lifted) graphical models

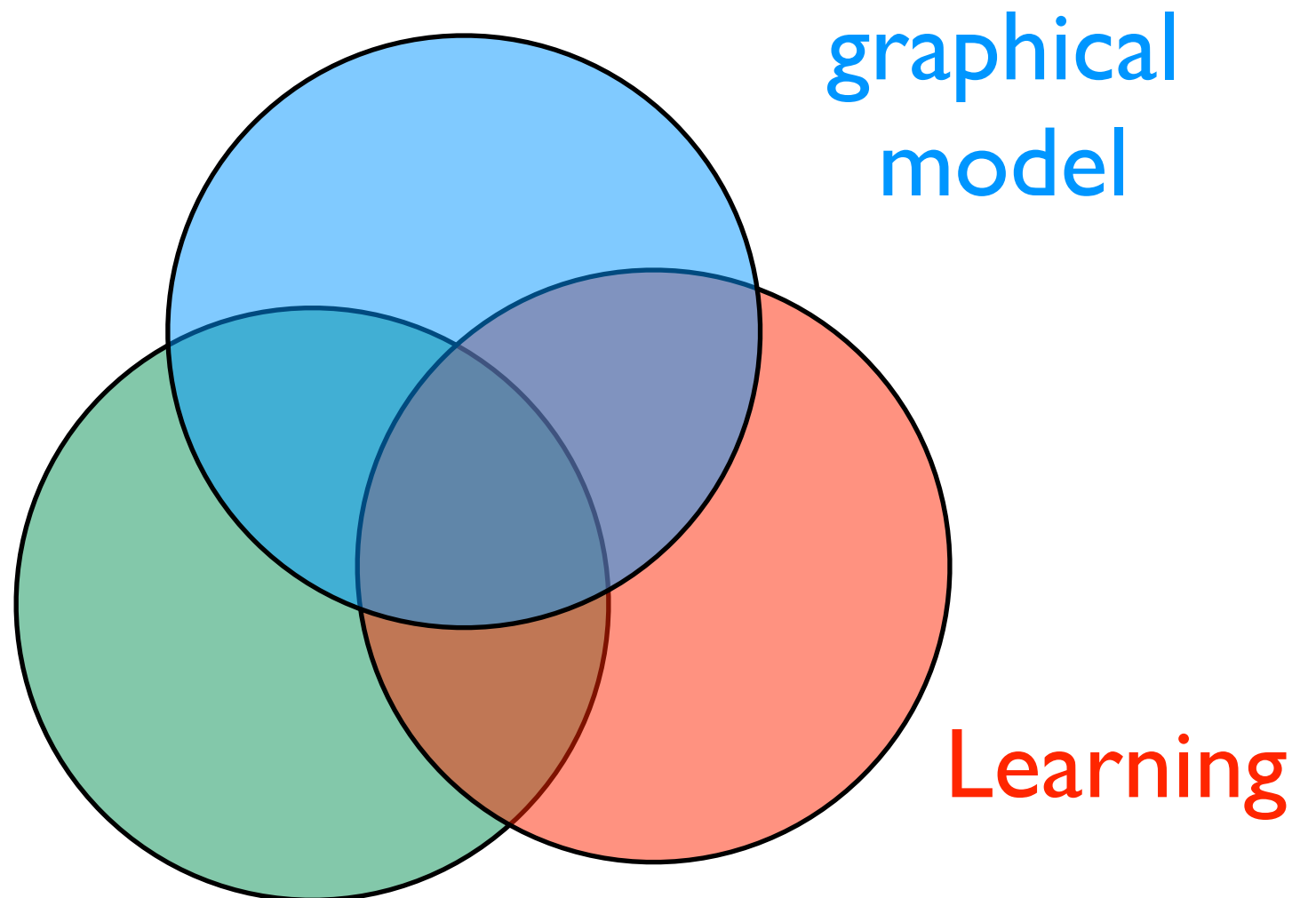
Lifted graphical models



Lifted graphical models

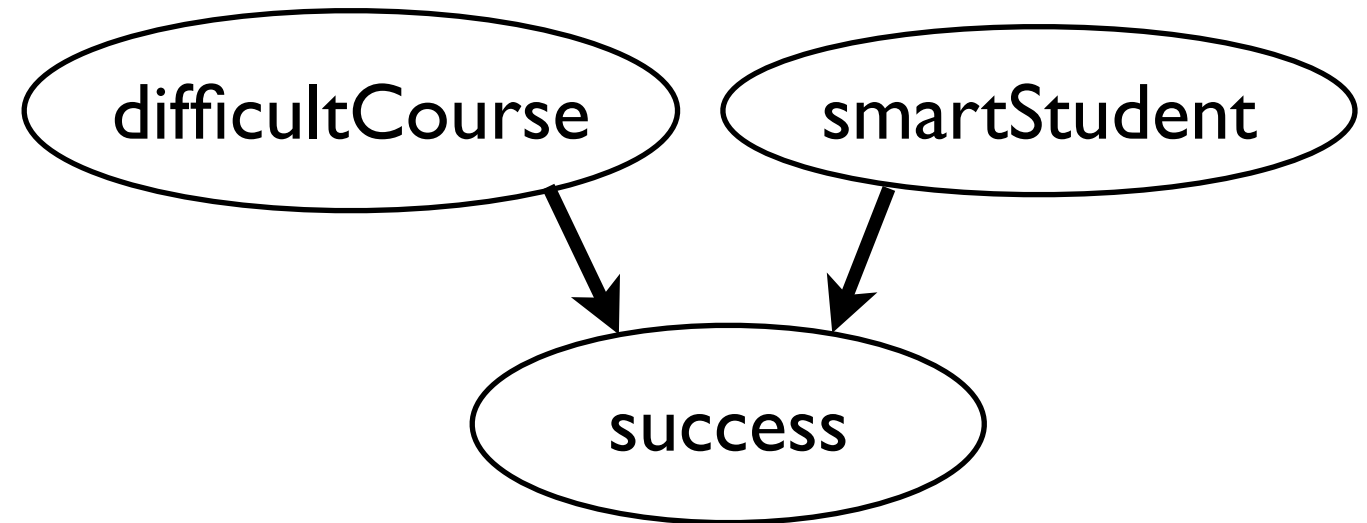


Reasoning with
relational data



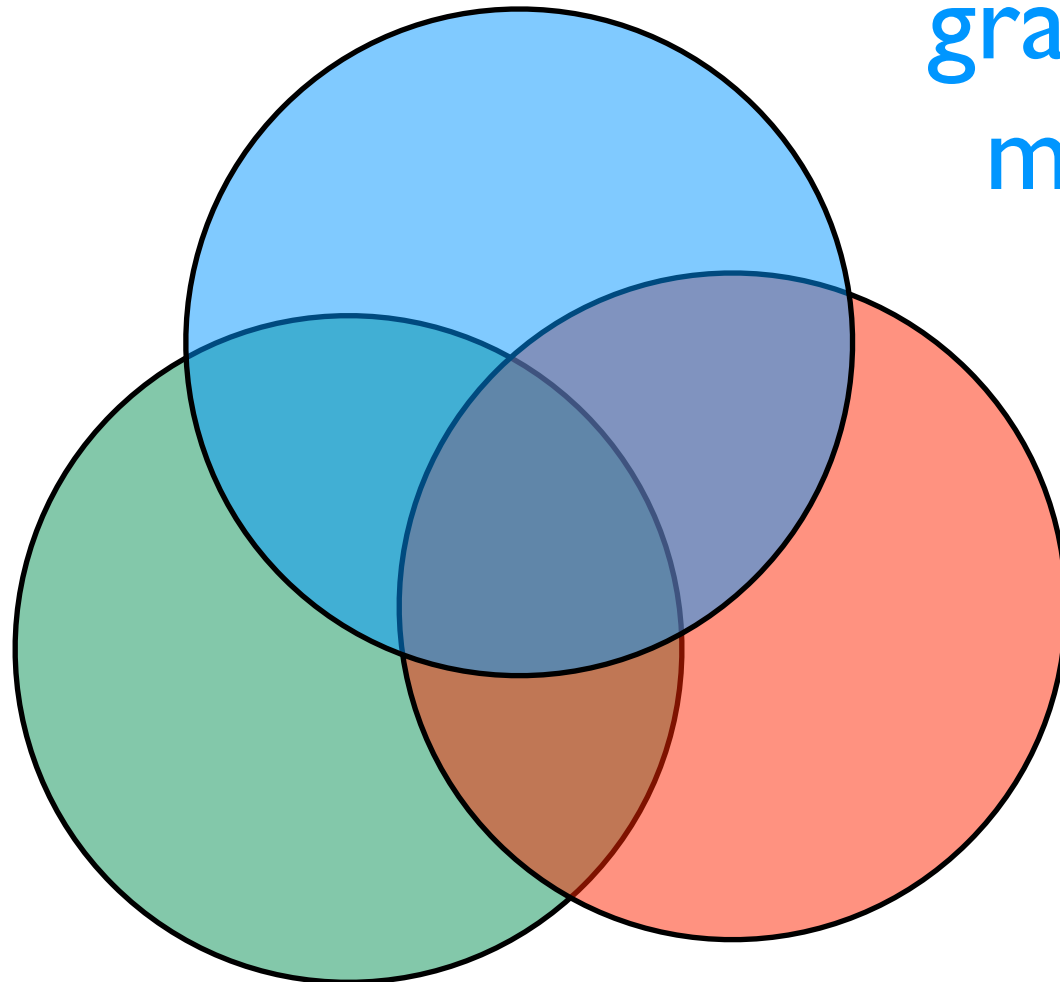
Lifted graphical models

fixed set of random variables



Reasoning with
relational data

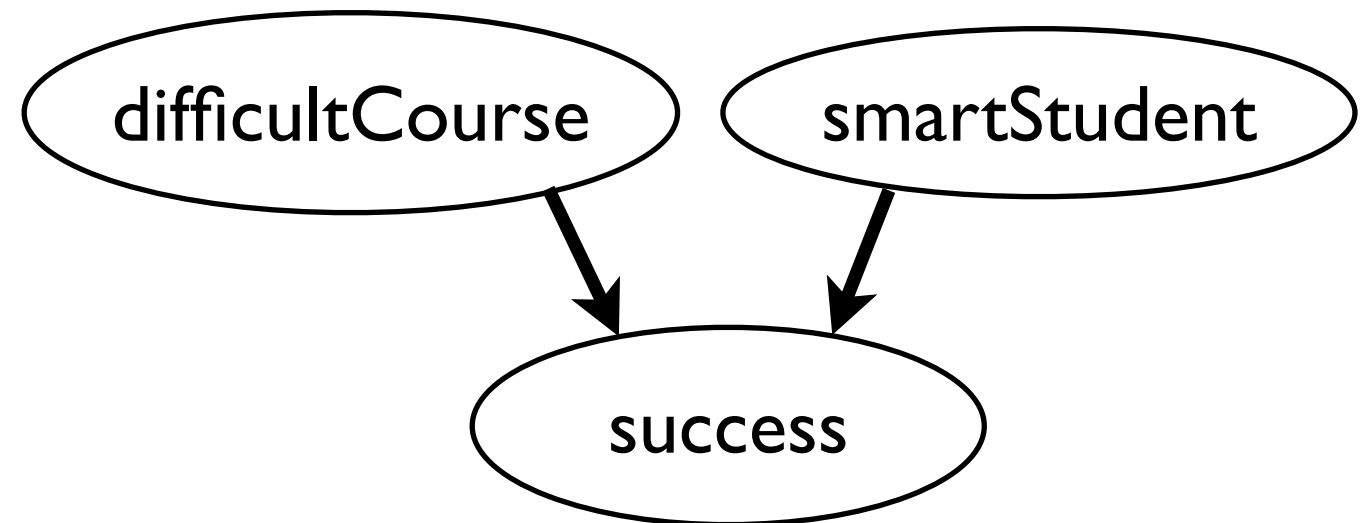
graphical
model



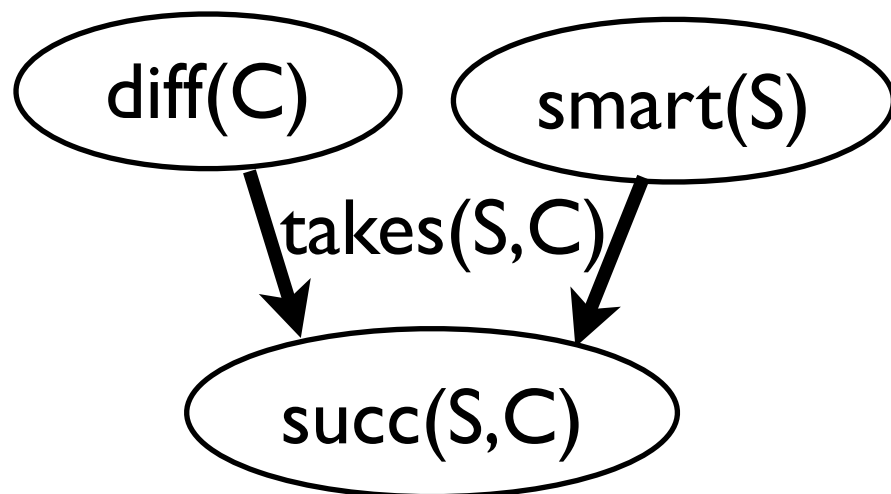
Learning

Lifted graphical models

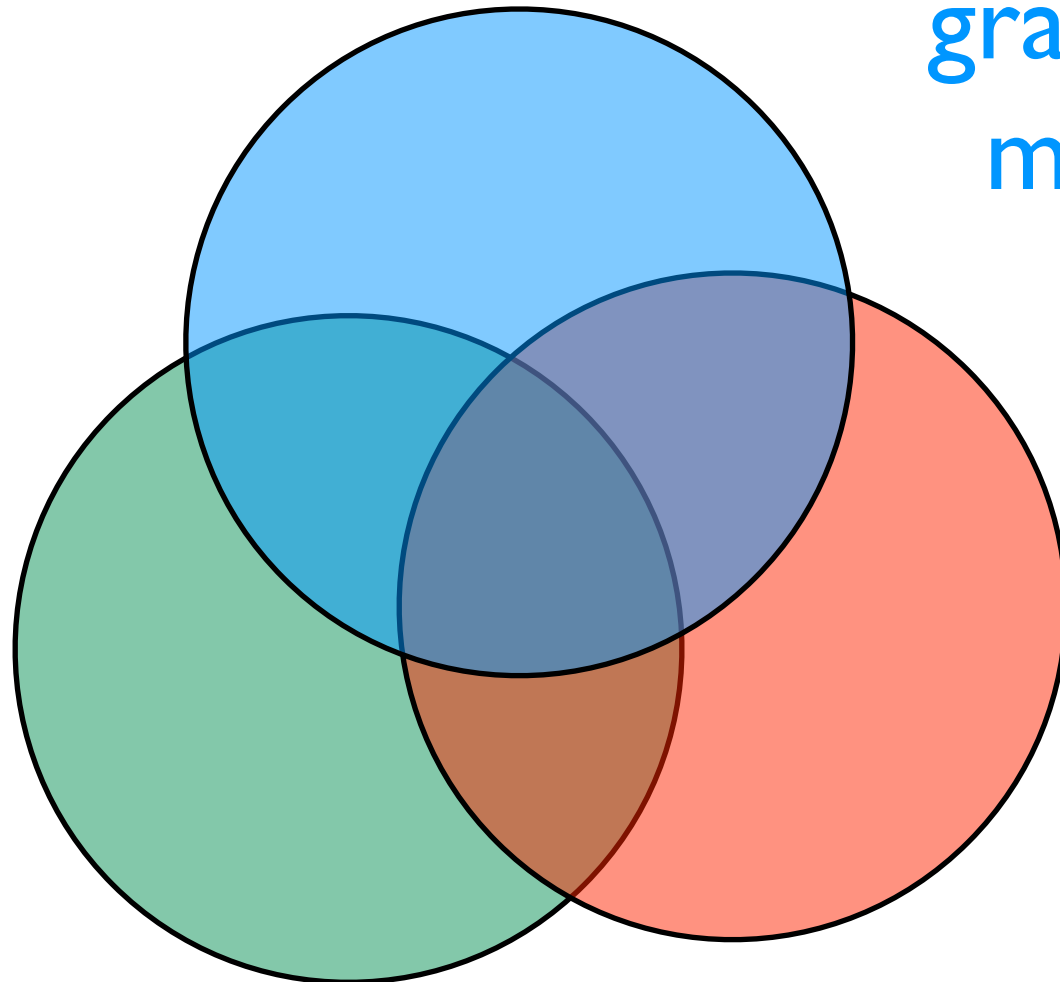
fixed set of random variables



**relational definition
of graphical model**



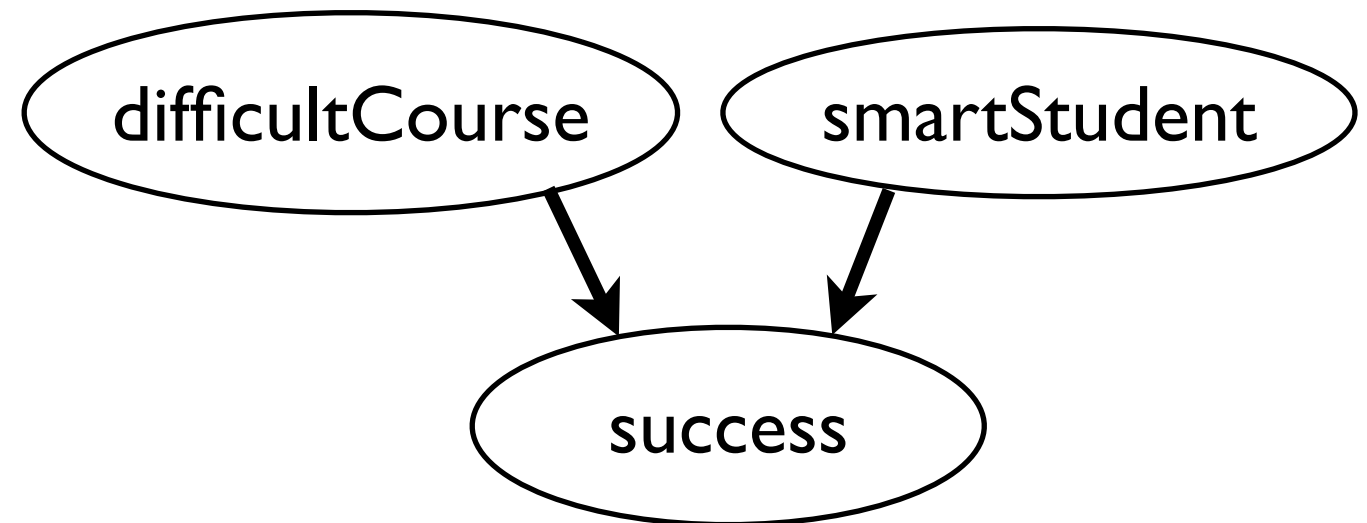
**graphical
model**



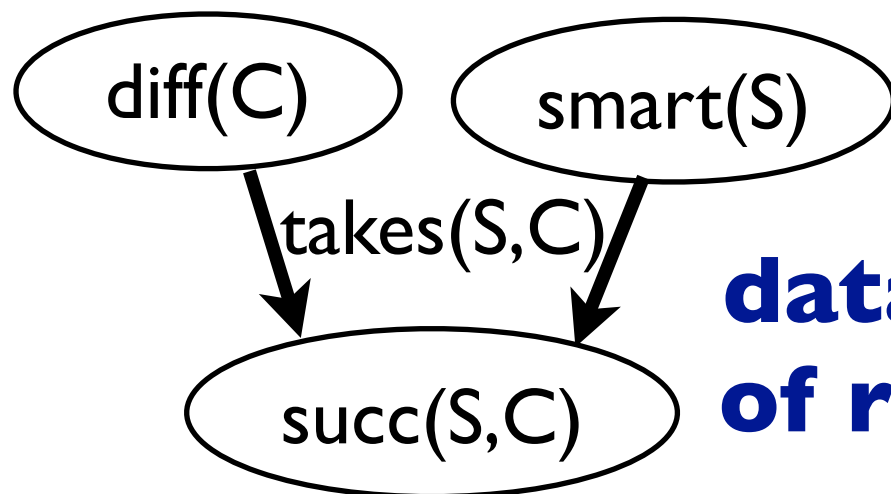
Learning

Lifted graphical models

fixed set of random variables

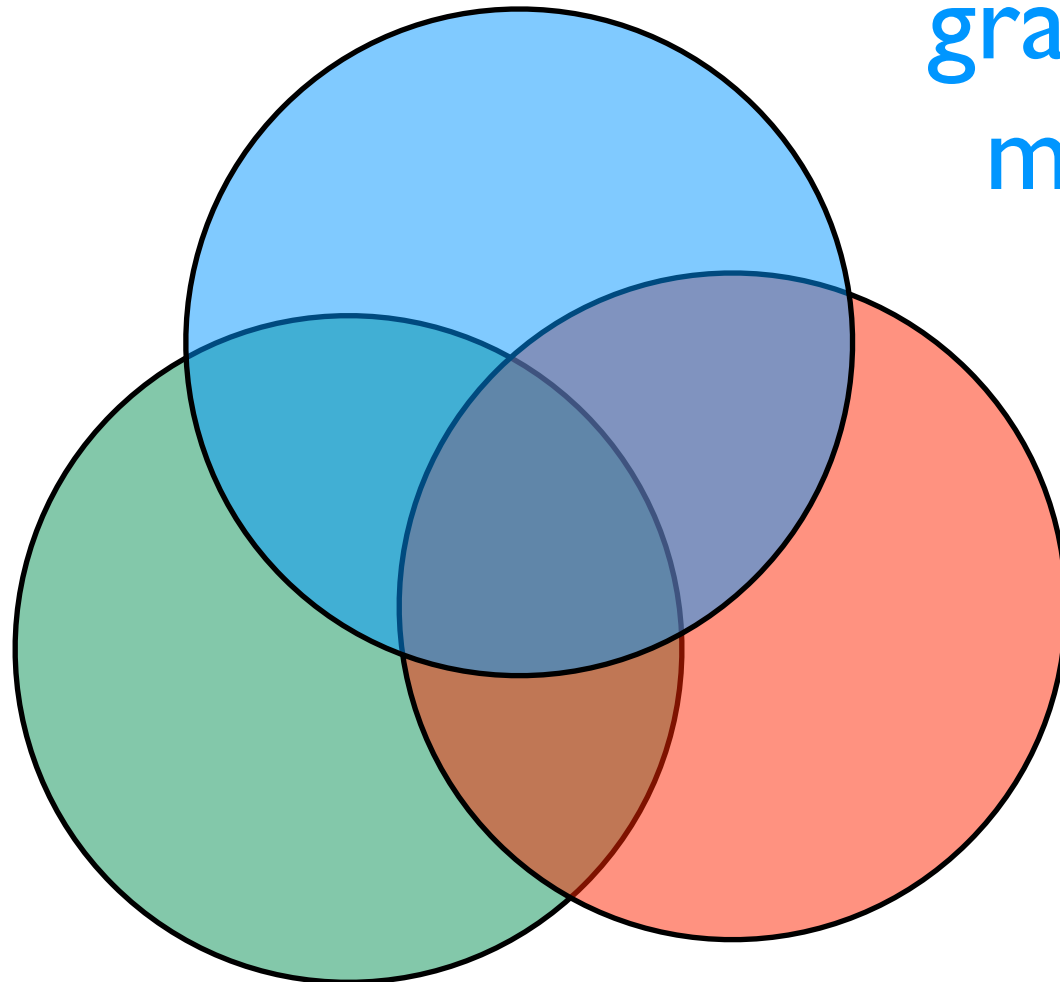


relational definition
of graphical model



**data-dependent set
of random variables**

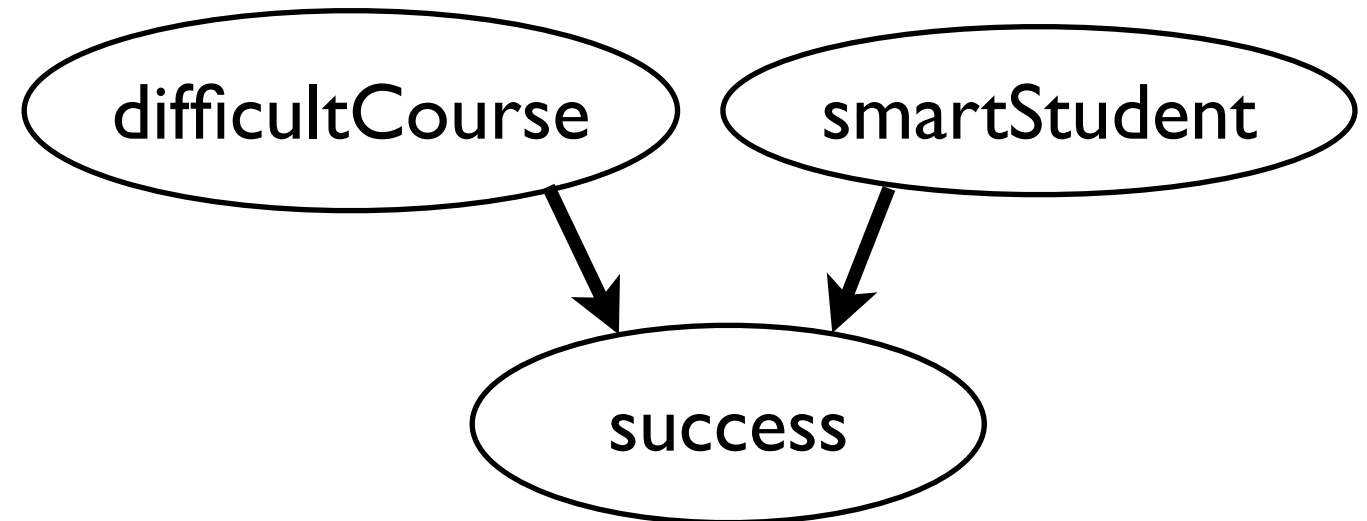
graphical
model



Learning

Lifted graphical models

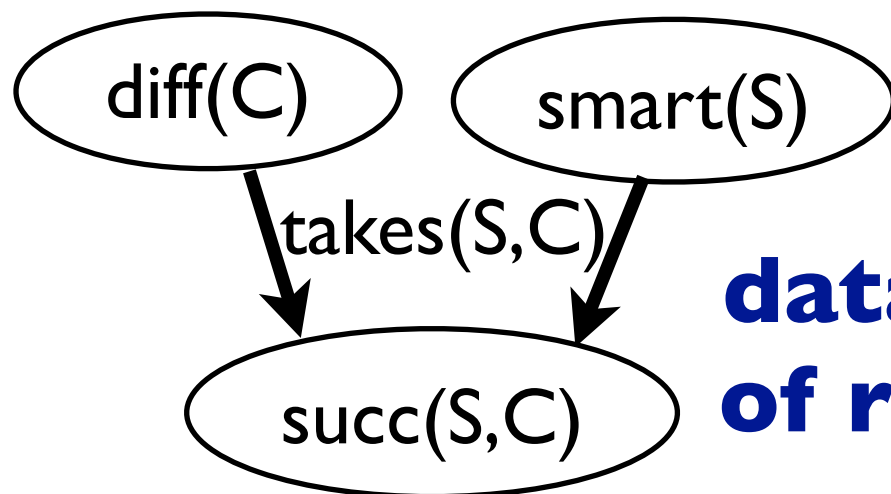
fixed set of random variables



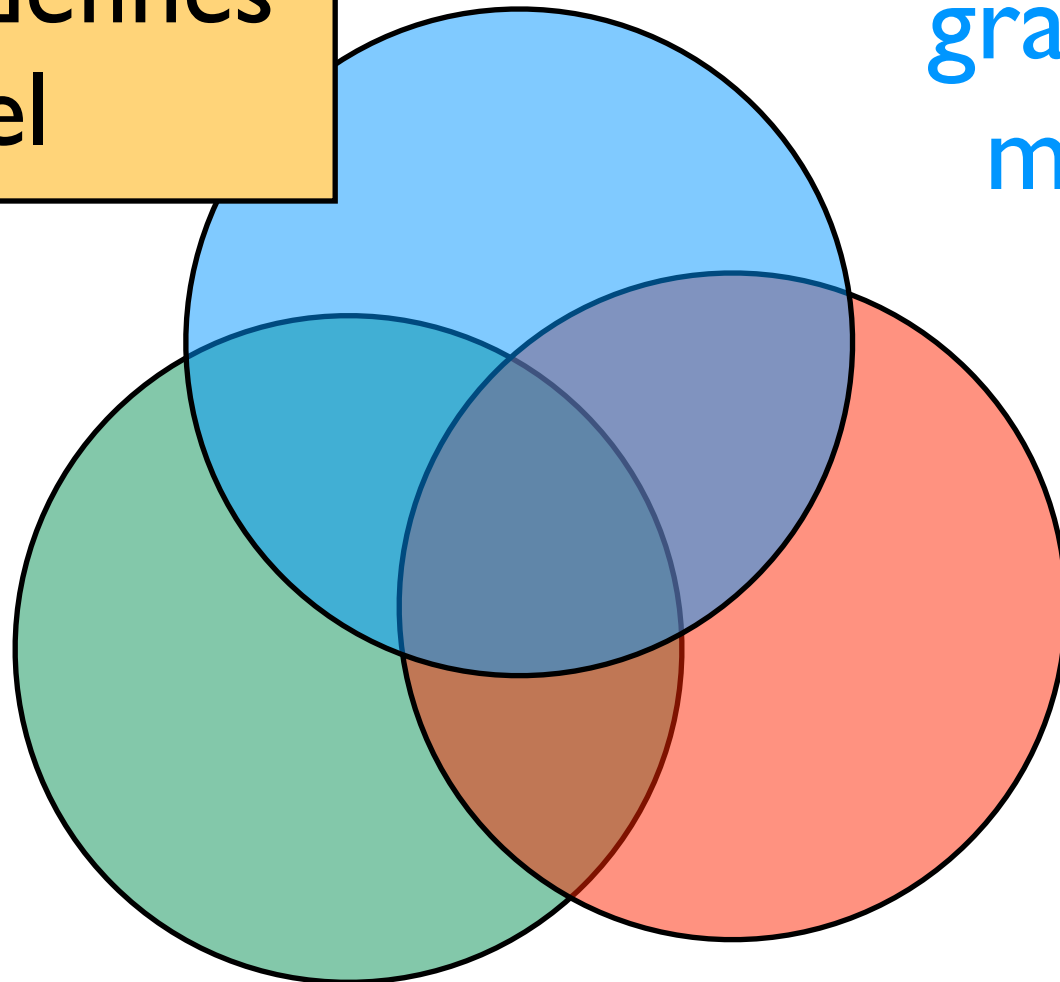
relational language defines graphical model

graphical model

relational definition of graphical model



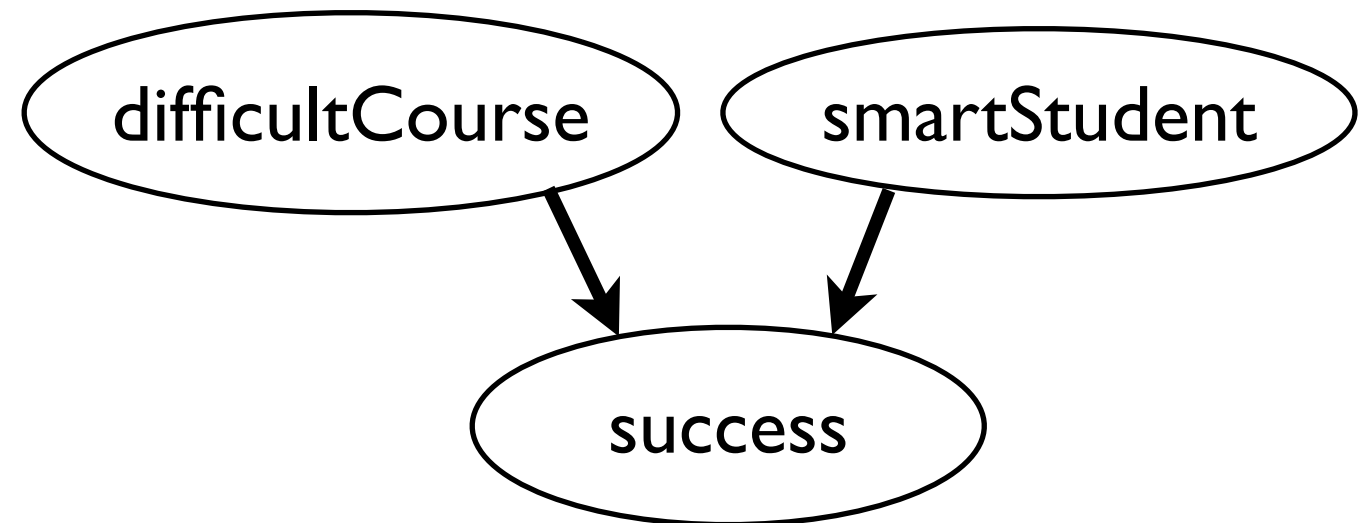
data-dependent set of random variables



Learning

Lifted graphical models

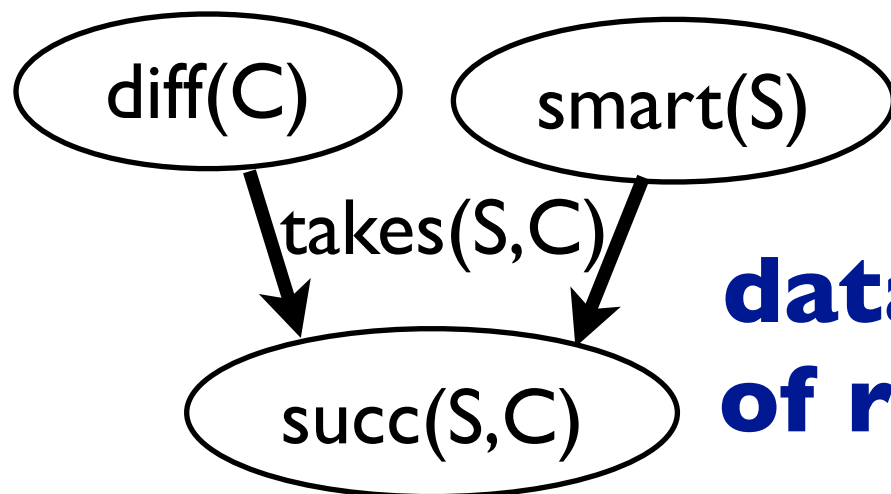
fixed set of random variables



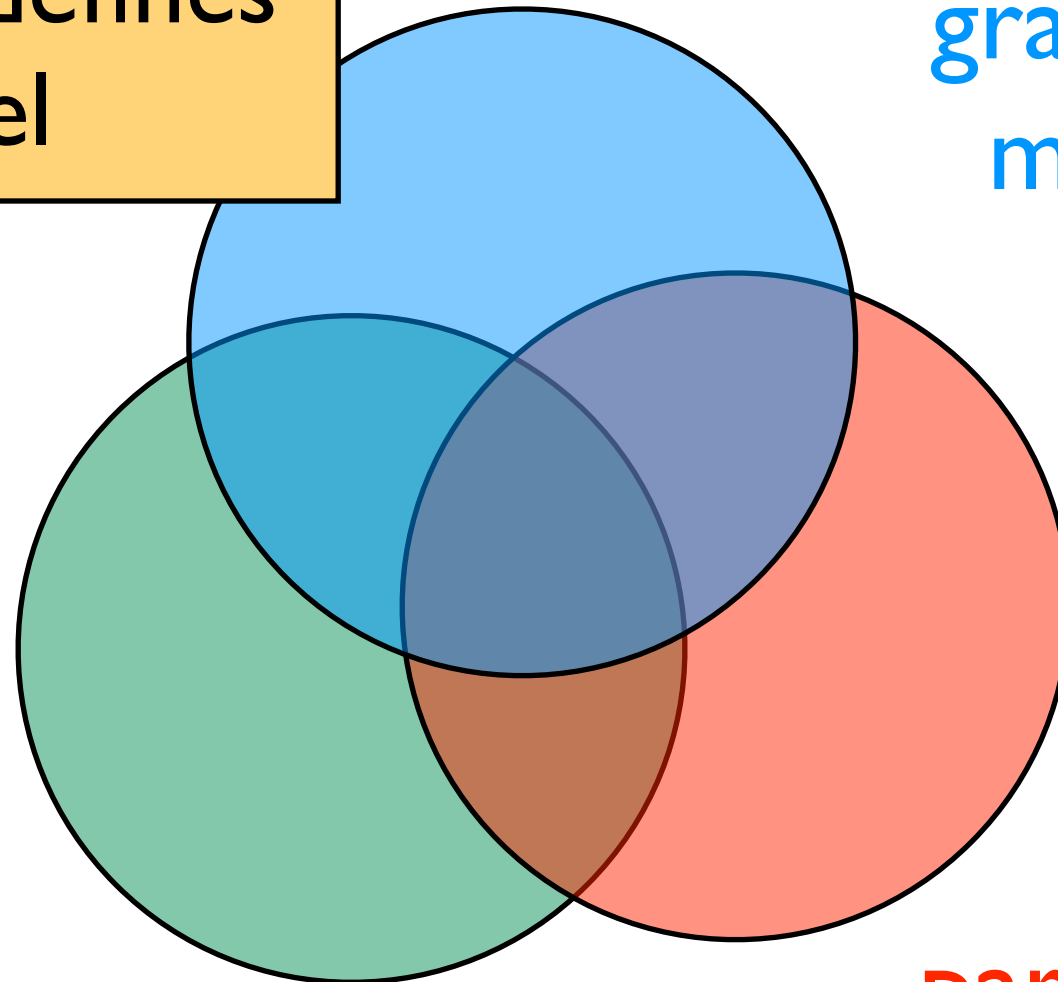
relational language defines graphical model

graphical model

relational definition of graphical model

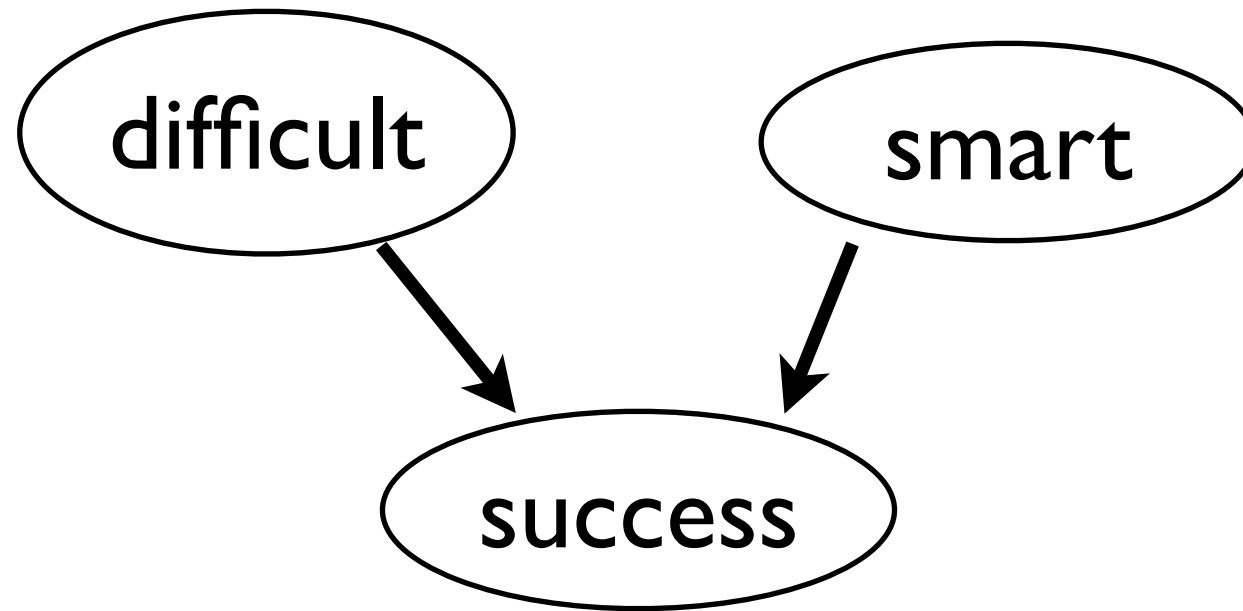


data-dependent set of random variables



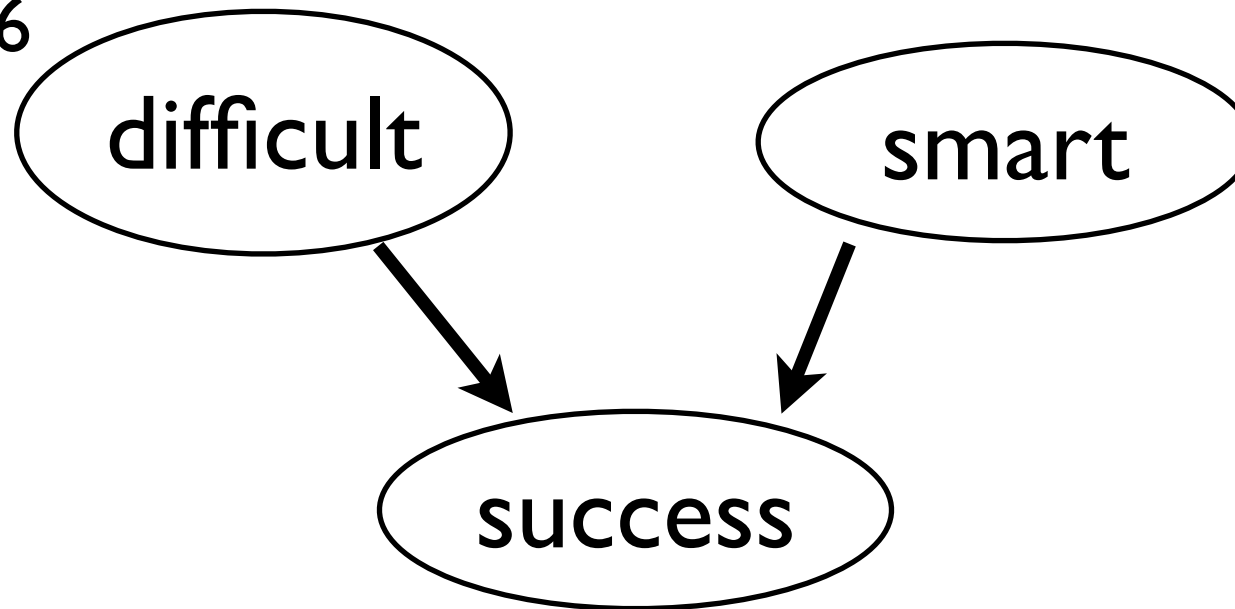
Learning parameters & structure

Bayesian Network



Bayesian Network

$$P(\text{difficult}=\text{T}) = 0.6$$



Bayesian Network

$$P(\text{difficult}=\text{T}) = 0.6$$

difficult

$$P(\text{smart}=\text{T}) = 0.7$$

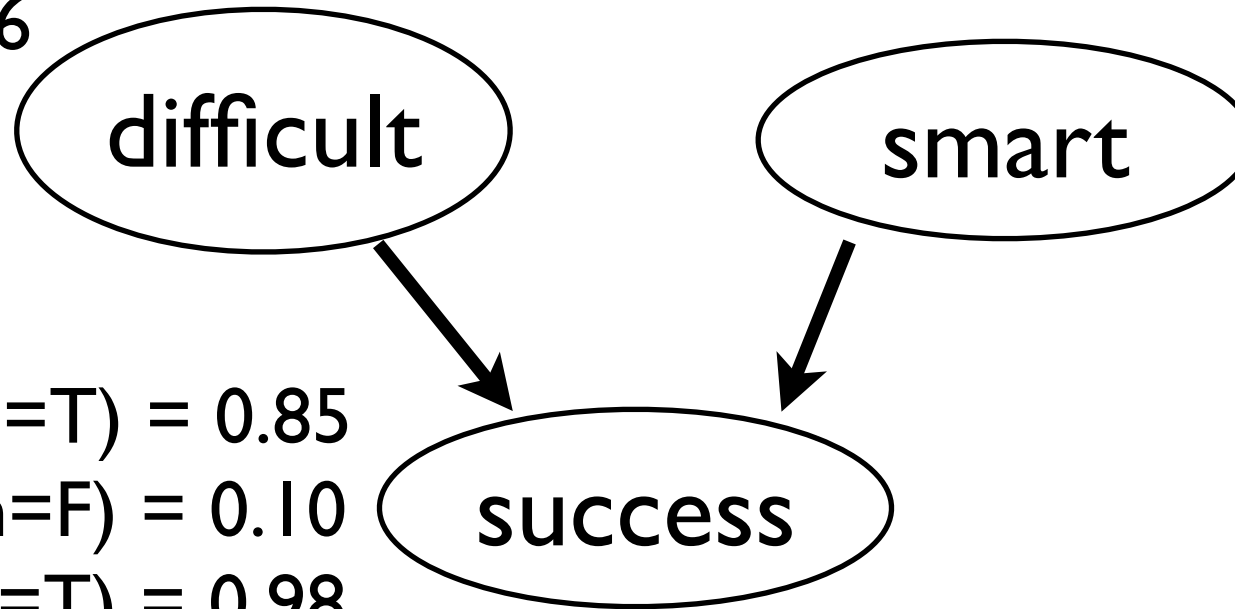
smart

success

Bayesian Network

$$P(\text{difficult}=\text{T}) = 0.6$$

$$P(\text{smart}=\text{T}) = 0.7$$



$$P(\text{success}=\text{T}|\text{d}=\text{T},\text{sm}=\text{T}) = 0.85$$

$$P(\text{success}=\text{T}|\text{d}=\text{T},\text{sm}=\text{F}) = 0.10$$

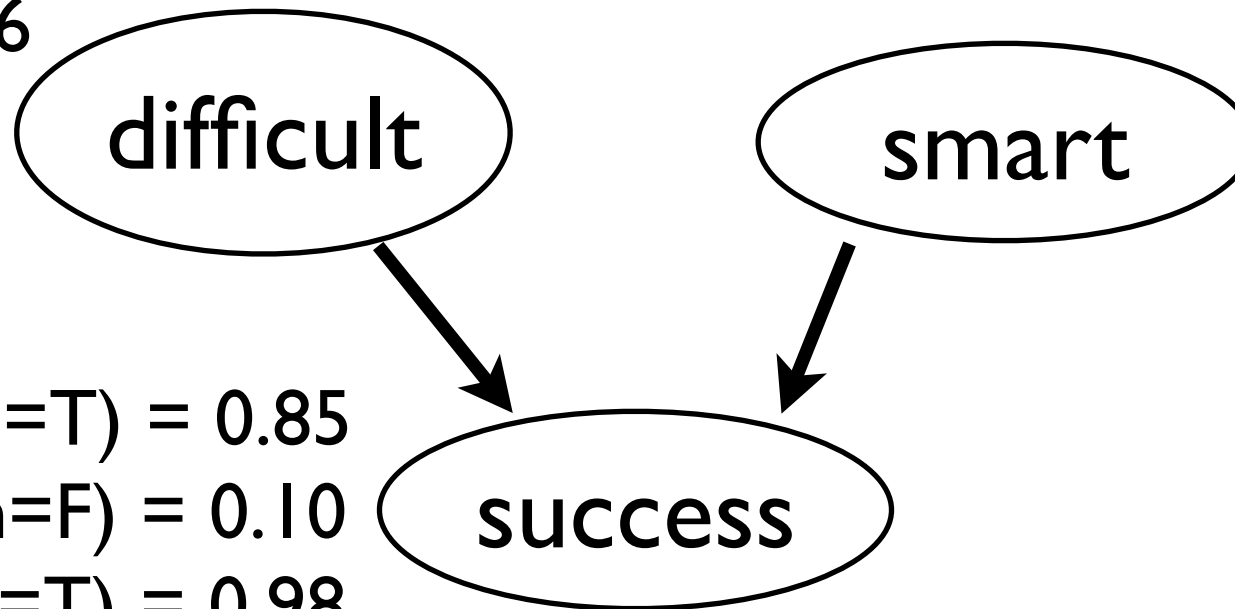
$$P(\text{success}=\text{T}|\text{d}=\text{F},\text{sm}=\text{T}) = 0.98$$

$$P(\text{success}=\text{T}|\text{d}=\text{F},\text{sm}=\text{F}) = 0.45$$

Bayesian Network

$$P(\text{difficult}=\text{T}) = 0.6$$

$$P(\text{smart}=\text{T}) = 0.7$$



$$P(\text{success}=\text{T}|\text{d}=\text{T},\text{sm}=\text{T}) = 0.85$$

$$P(\text{success}=\text{T}|\text{d}=\text{T},\text{sm}=\text{F}) = 0.10$$

$$P(\text{success}=\text{T}|\text{d}=\text{F},\text{sm}=\text{T}) = 0.98$$

$$P(\text{success}=\text{T}|\text{d}=\text{F},\text{sm}=\text{F}) = 0.45$$

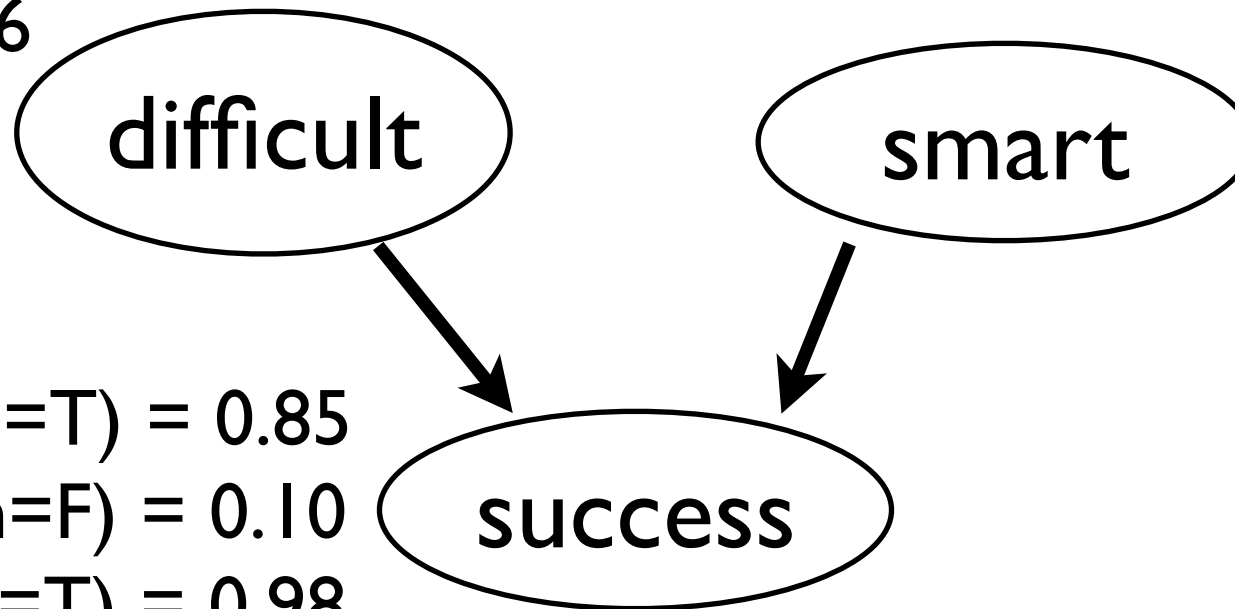
joint distribution

$$P(\text{difficult}) \times P(\text{smart}) \times P(\text{success}|\text{difficult},\text{smart})$$

Bayesian Network

$$P(\text{difficult}=\text{T}) = 0.6$$

$$P(\text{smart}=\text{T}) = 0.7$$



$$P(\text{success}=\text{T}|\text{d}=\text{T},\text{sm}=\text{T}) = 0.85$$

$$P(\text{success}=\text{T}|\text{d}=\text{T},\text{sm}=\text{F}) = 0.10$$

$$P(\text{success}=\text{T}|\text{d}=\text{F},\text{sm}=\text{T}) = 0.98$$

$$P(\text{success}=\text{T}|\text{d}=\text{F},\text{sm}=\text{F}) = 0.45$$

joint distribution

$$P(\text{difficult}) \times P(\text{smart}) \times P(\text{success}|\text{difficult},\text{smart})$$

directed
acyclic graph

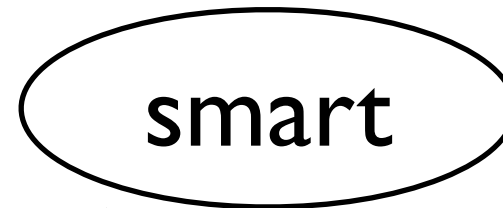
$$P(X_1, \dots, X_n) = \prod_{i=1, \dots, n} P(X_i | \text{par}(X_i))$$

Bayesian Network in ProbLog?

$$P(\text{difficult}=\text{T}) = 0.6$$



$$P(\text{smart}=\text{T}) = 0.7$$



$$P(\text{success}=\text{T}|\text{d}=\text{T},\text{sm}=\text{T}) = 0.85$$

$$P(\text{success}=\text{T}|\text{d}=\text{T},\text{sm}=\text{F}) = 0.10$$

$$P(\text{success}=\text{T}|\text{d}=\text{F},\text{sm}=\text{T}) = 0.98$$

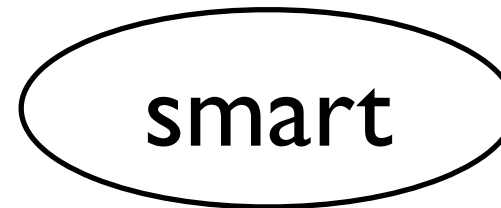
$$P(\text{success}=\text{T}|\text{d}=\text{F},\text{sm}=\text{F}) = 0.45$$

Bayesian Network in ProbLog?

$$P(\text{difficult}=\text{T}) = 0.6$$



$$P(\text{smart}=\text{T}) = 0.7$$



$$P(\text{success}=\text{T}|\text{d}=\text{T},\text{sm}=\text{T}) = 0.85$$

$$P(\text{success}=\text{T}|\text{d}=\text{T},\text{sm}=\text{F}) = 0.10$$

$$P(\text{success}=\text{T}|\text{d}=\text{F},\text{sm}=\text{T}) = 0.98$$

$$P(\text{success}=\text{T}|\text{d}=\text{F},\text{sm}=\text{F}) = 0.45$$

```
0.6::difficult.
```

```
0.7::smart.
```

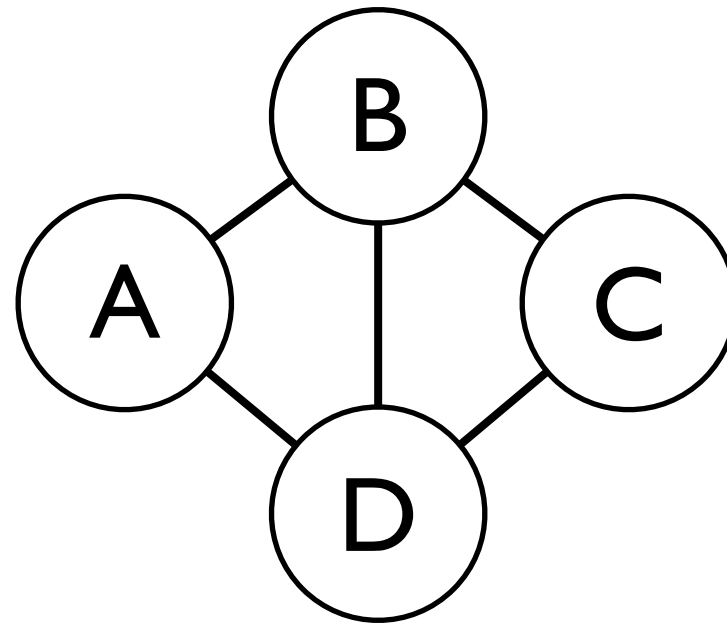
```
0.85::success <- difficult, smart.
```

```
0.10::success <- difficult, \+smart.
```

```
0.98::success <- \+difficult, smart.
```

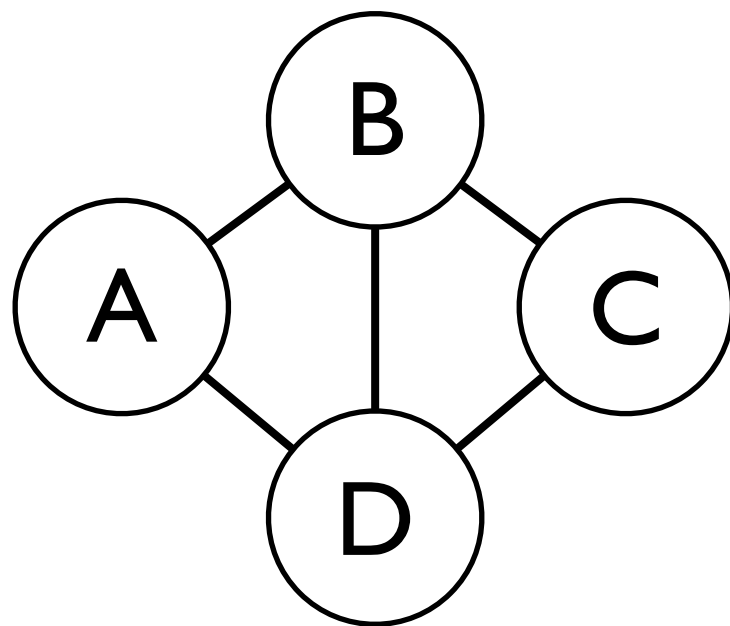
```
0.45::success <- \+difficult, \+smart.
```

Markov Network



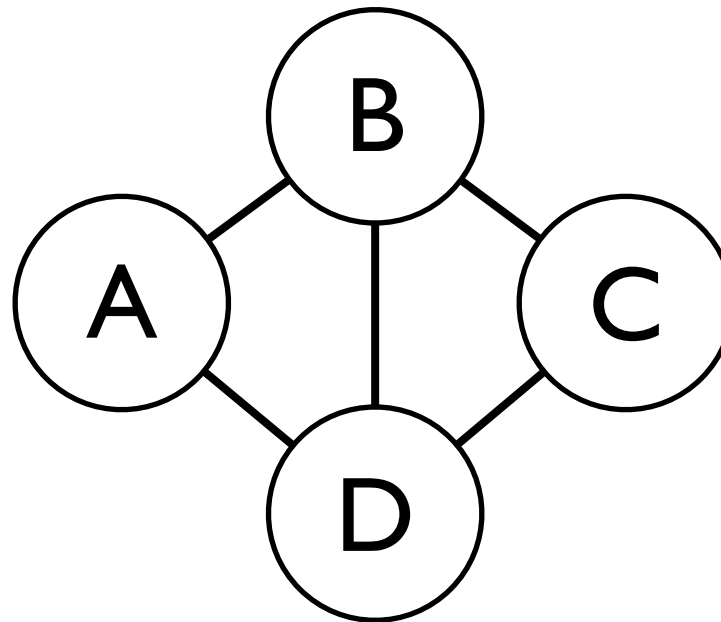
Markov Network

A	B	D	$\phi_1(A, B, D)$
T	T	T	5
T	T	F	1
T	F	T	1
T	F	F	5
F	T	T	1
F	T	F	5
F	F	T	5
F	F	F	1



Markov Network

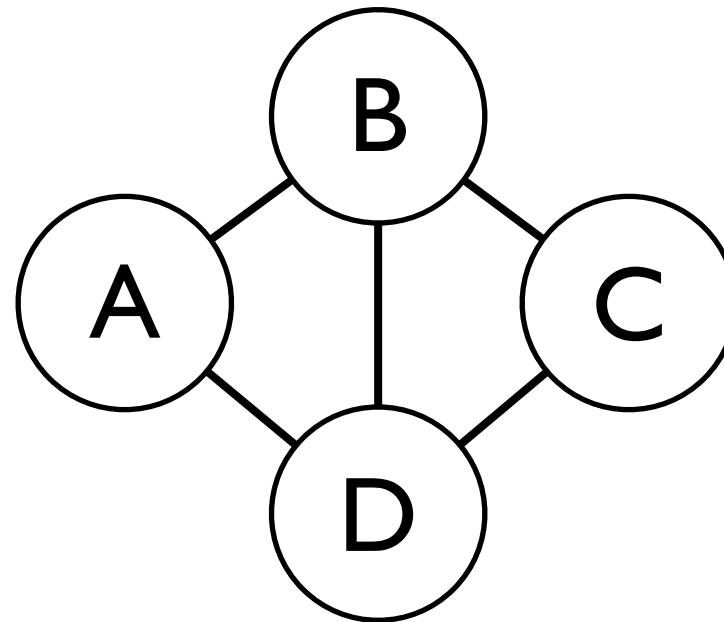
A	B	D	$\phi_1(A, B, D)$
T	T	T	5
T	T	F	1
T	F	T	1
T	F	F	5
F	T	T	1
F	T	F	5
F	F	T	5
F	F	F	1



B	C	D	$\phi_2(B, C, D)$
T	T	T	4
T	T	F	2
T	F	T	3
T	F	F	1
F	T	T	2
F	T	F	4
F	F	T	1
F	F	F	3

Markov Network

A	B	D	$\phi_1(A, B, D)$
T	T	T	5
T	T	F	1
T	F	T	1
T	F	F	5
F	T	T	1
F	T	F	5
F	F	T	5
F	F	F	1

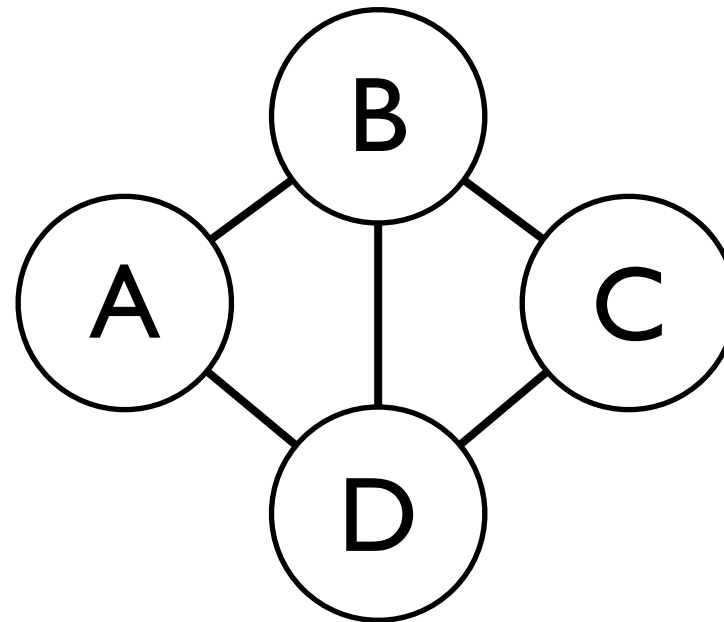


B	C	D	$\phi_2(B, C, D)$
T	T	T	4
T	T	F	2
T	F	T	3
T	F	F	1
F	T	T	2
F	T	F	4
F	F	T	1
F	F	F	3

$$\begin{aligned}
 P(A = T, B = F, C = T, D = F) &= \frac{1}{Z} \phi_1(A = T, B = F, D = F) \phi_2(B = F, C = T, D = F) \\
 &= \frac{1}{Z} \cdot 5 \cdot 4
 \end{aligned}$$

Markov Network

A	B	D	$\phi_1(A, B, D)$
T	T	T	5
T	T	F	1
T	F	T	1
T	F	F	5
F	T	T	1
F	T	F	5
F	F	T	5
F	F	F	1



B	C	D	$\phi_2(B, C, D)$
T	T	T	4
T	T	F	2
T	F	T	3
T	F	F	1
F	T	T	2
F	T	F	4
F	F	T	1
F	F	F	3

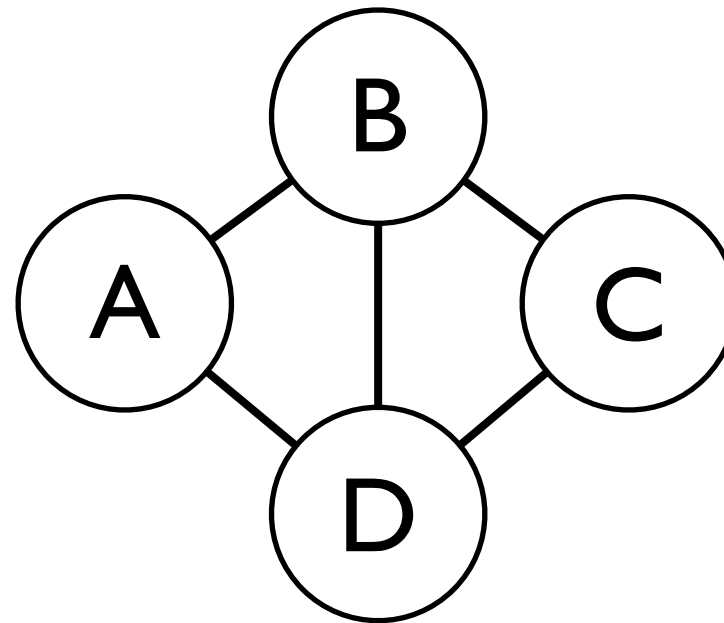
$$\begin{aligned}
 P(A = T, B = F, C = T, D = F) &= \frac{1}{Z} \phi_1(A = T, B = F, D = F) \phi_2(B = F, C = T, D = F) \\
 &= \frac{1}{Z} \cdot 5 \cdot 4
 \end{aligned}$$

undirected graph

$$\begin{aligned}
 P(\mathbf{X} = \mathbf{x}) &= \frac{1}{Z} \prod_{\phi_i \in F} \phi_i(\mathbf{X}_{\phi_i} = \mathbf{x}_{\phi_i}) \\
 Z &= \sum_{\mathbf{x}'} \prod_{\phi_i \in F} \phi_i(\mathbf{X}_{\phi_i} = \mathbf{x}'_{\phi_i})
 \end{aligned}$$

Markov Network in ProbLog?

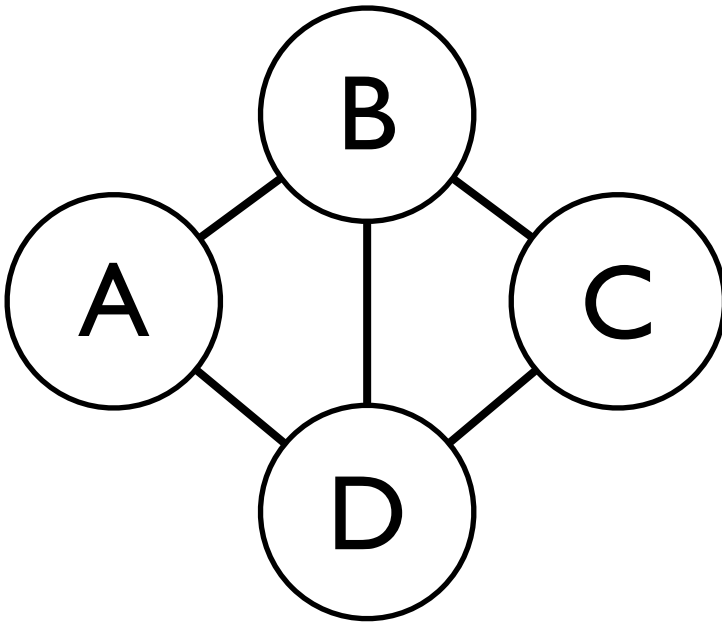
A	B	D	$\phi_1(A, B, D)$
T	T	T	5
T	T	F	1
T	F	T	1
T	F	F	5
F	T	T	1
F	T	F	5
F	F	T	5
F	F	F	1



B	C	D	$\phi_2(B, C, D)$
T	T	T	4
T	T	F	2
T	F	T	3
T	F	F	1
F	T	T	2
F	T	F	4
F	F	T	1
F	F	F	3

Markov Network in ProbLog?

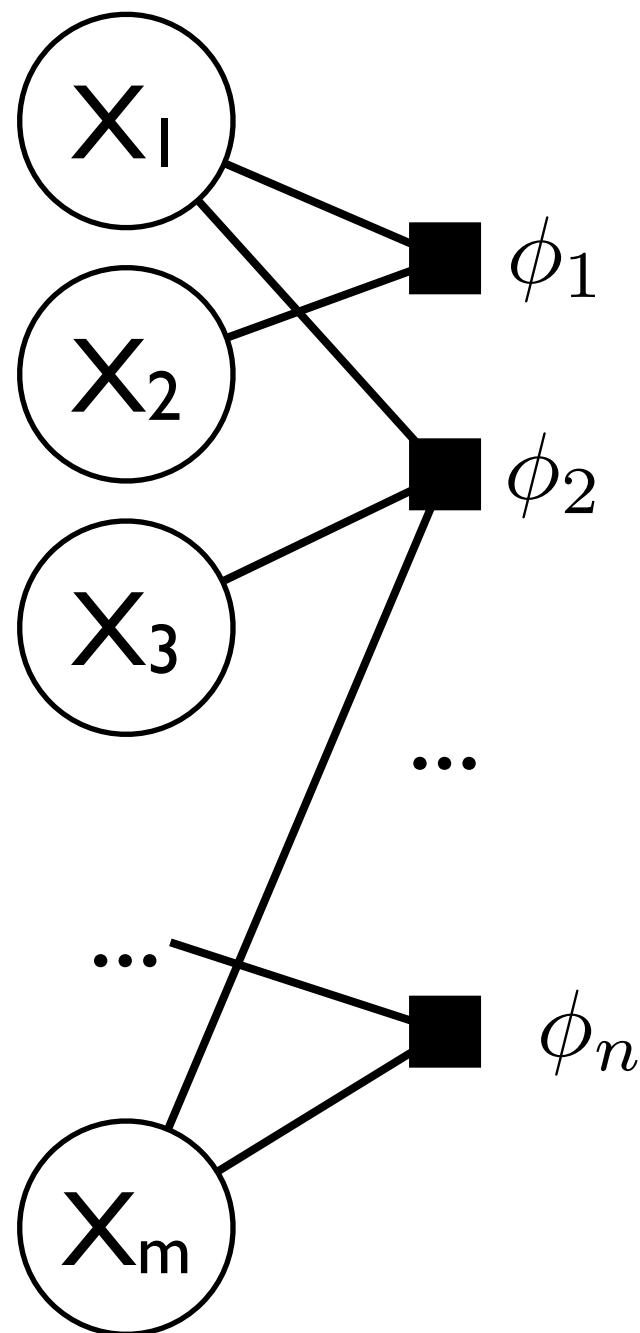
A	B	D	$\phi_1(A, B, D)$
T	T	T	5
T	T	F	1
T	F	T	1
T	F	F	5
F	T	T	1
F	T	F	5
F	F	T	5
F	F	F	1



B	C	D	$\phi_2(B, C, D)$
T	T	T	4
T	T	F	2
T	F	T	3
T	F	F	1
F	T	T	2
F	T	F	4
F	F	T	1
F	F	F	3

no direct mapping due to general
potential functions and normalization

Factor Graph

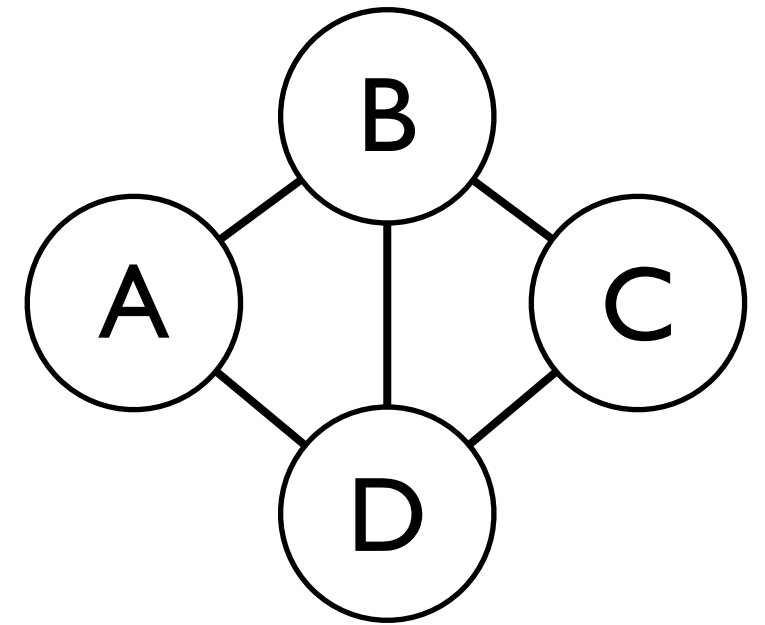


- bi-partite undirected graph
- variables X_1, \dots, X_m
- factors ϕ_1, \dots, ϕ_n map interpretations of subsets of variables to non-negative reals
- joint distribution

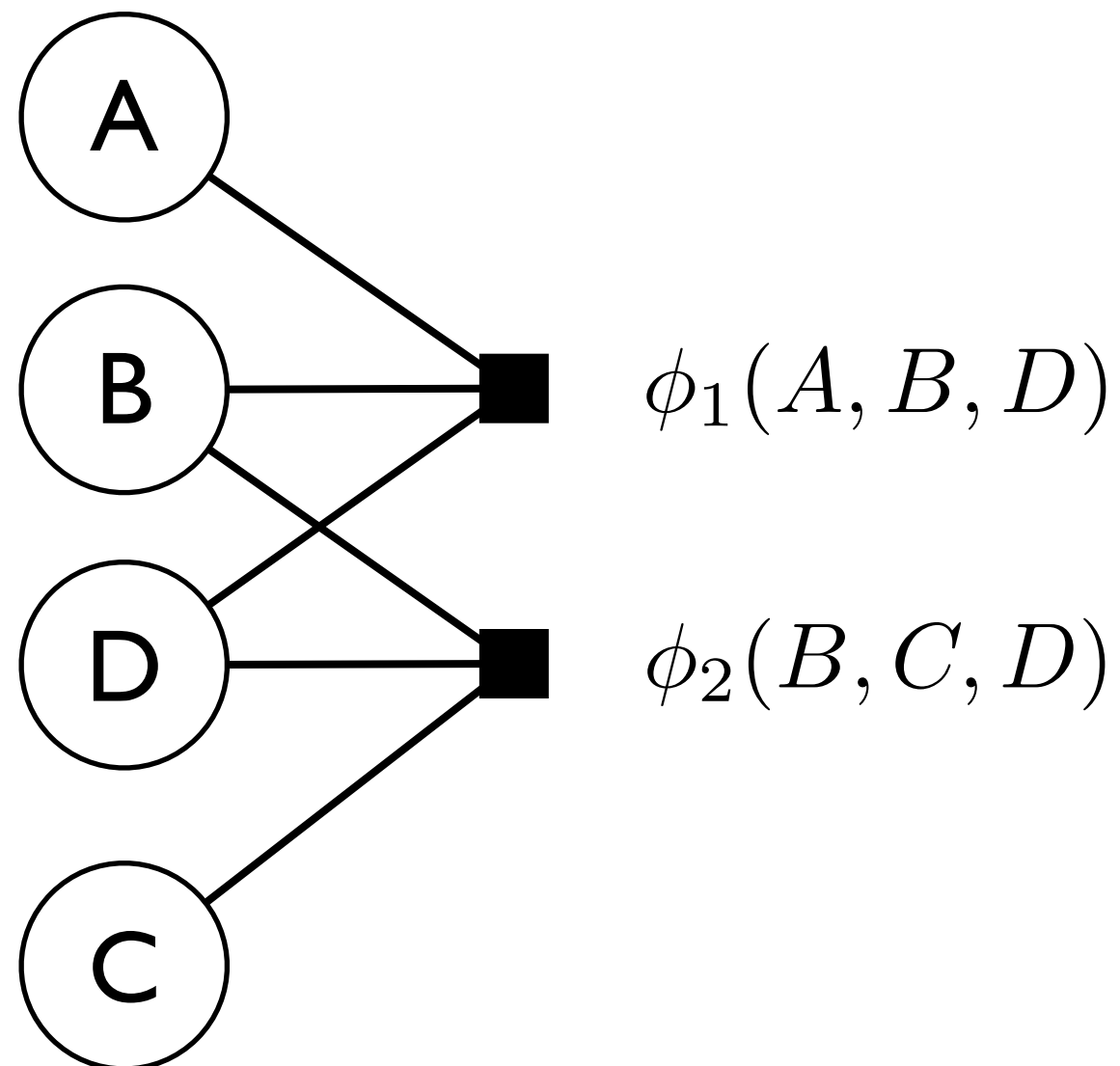
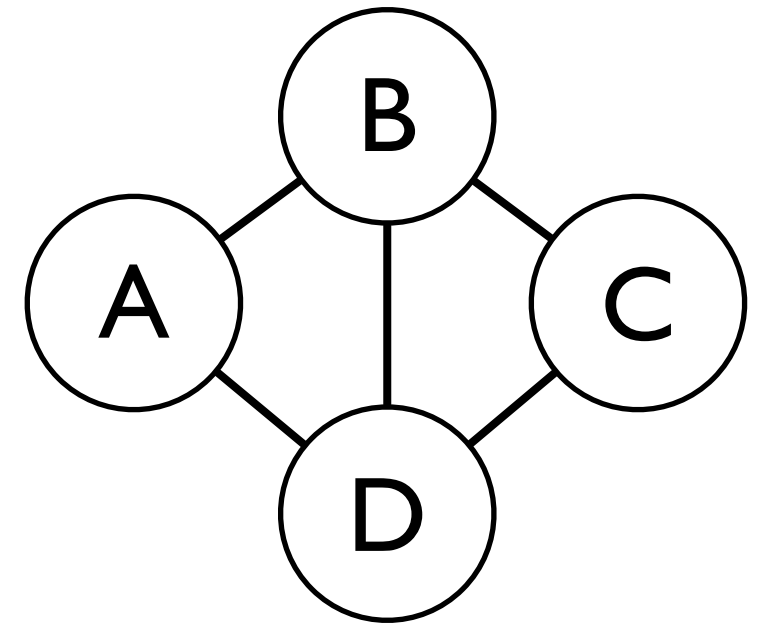
$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_{i=1..n} \phi_i(\mathbf{X}_{\phi_i} = \mathbf{x}_{\phi_i})$$

$$Z = \sum_{\mathbf{x}'} \prod_{i=1..n} \phi_i(\mathbf{X}_{\phi_i} = \mathbf{x}'_{\phi_i})$$

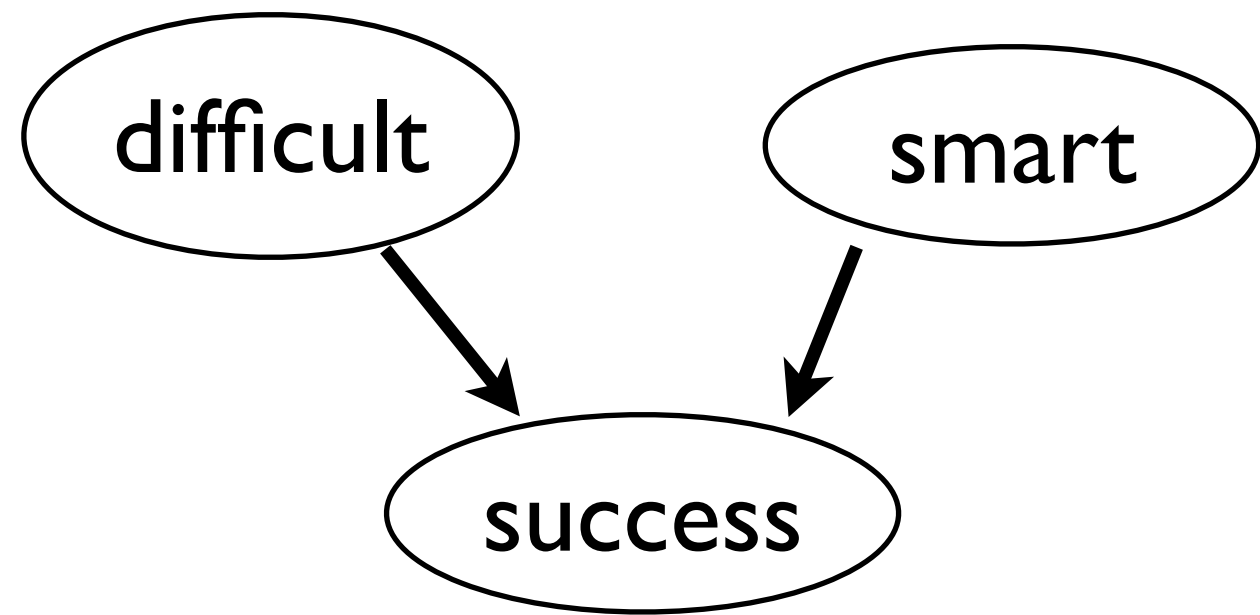
MN as Factor Graph?



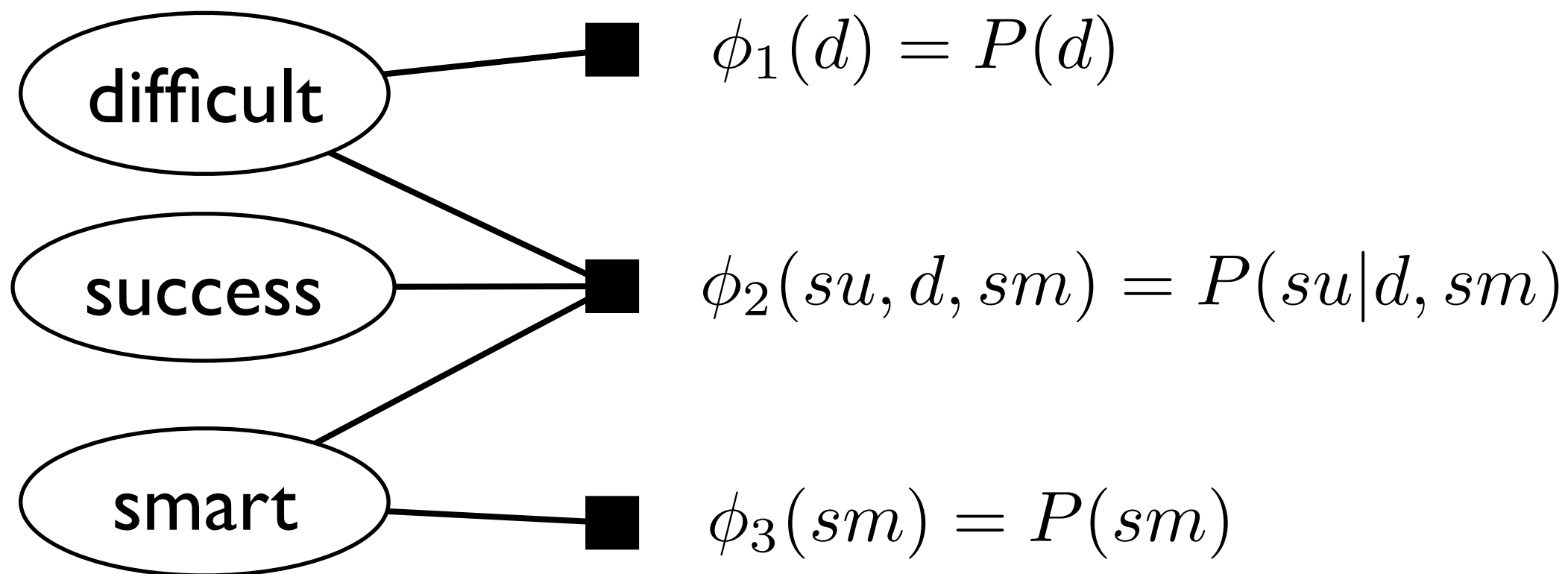
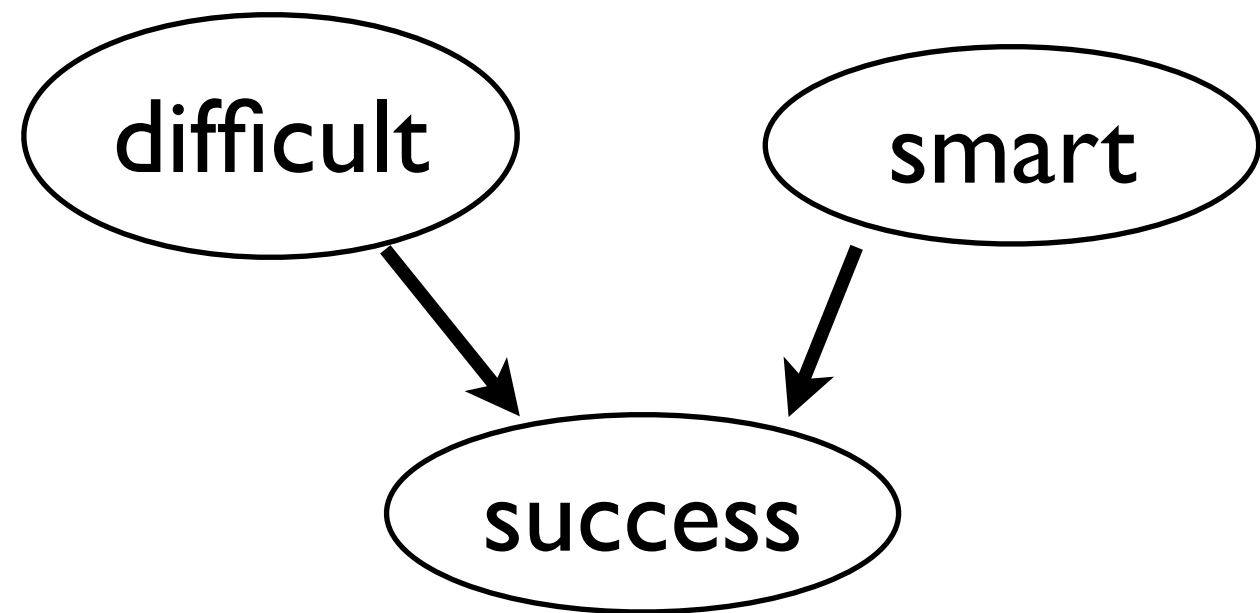
MN as Factor Graph?



BN as Factor Graph?

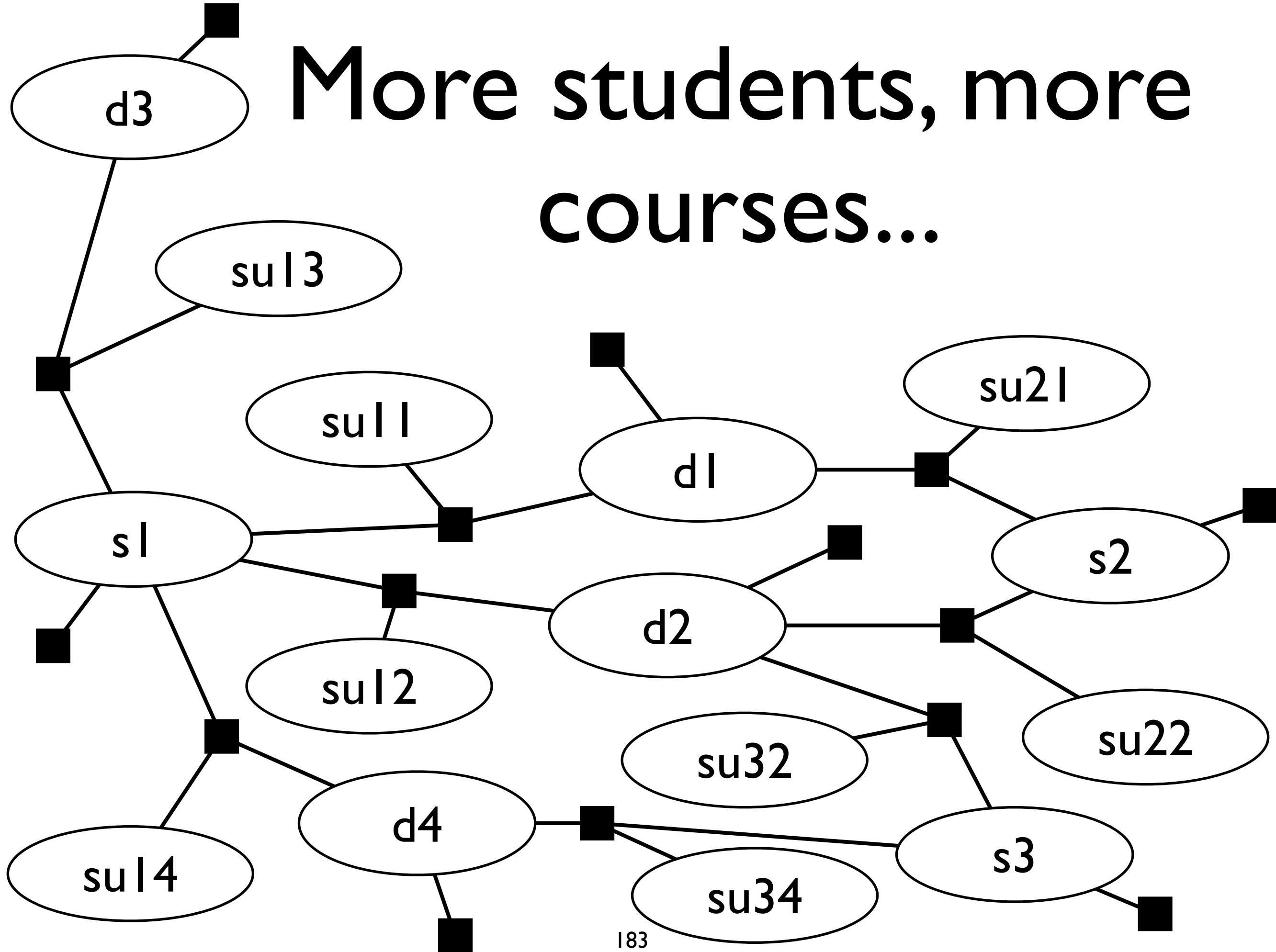


BN as Factor Graph?

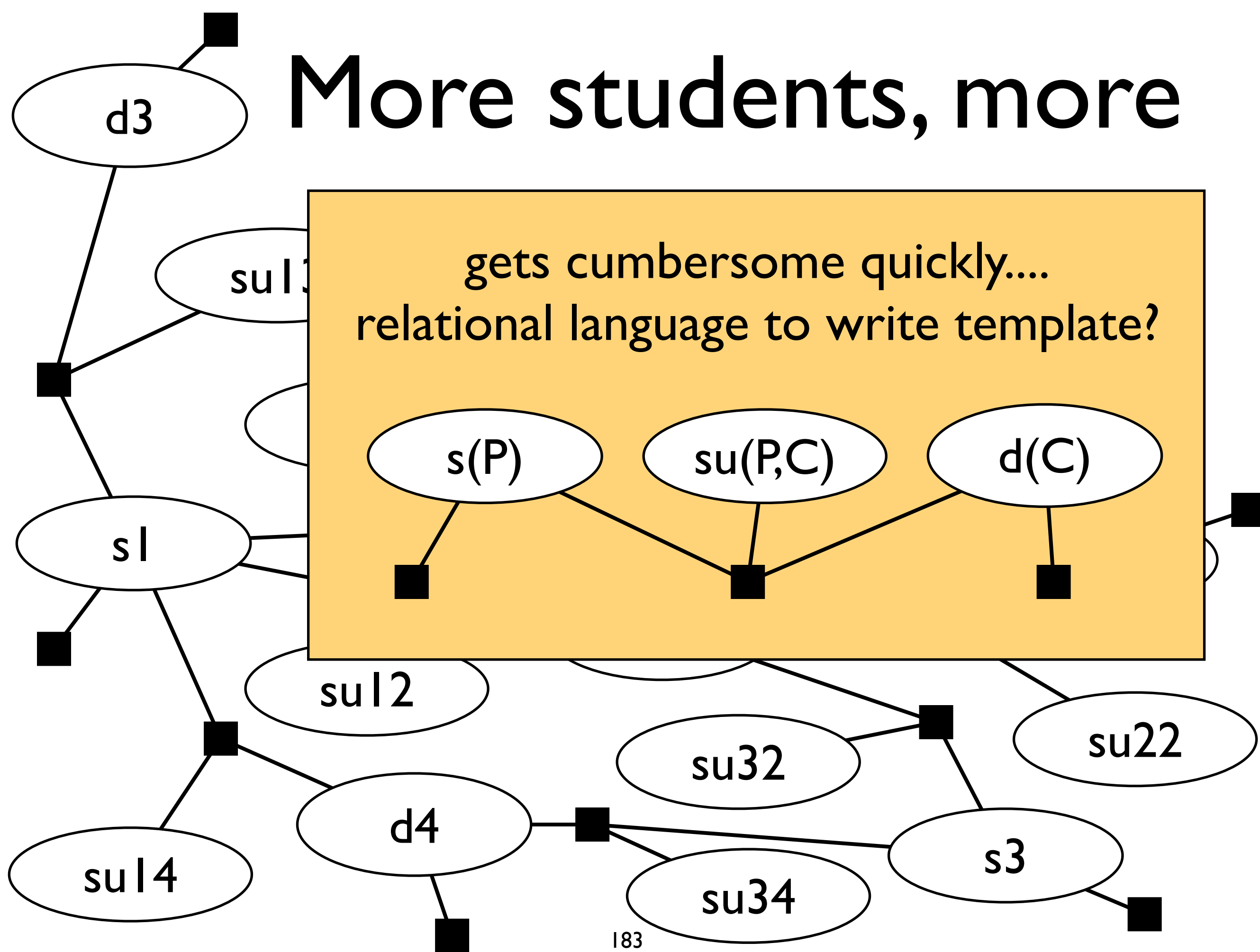


Z=1

More students, more courses...



More students, more



Lots of proposals in the literature, e.g.

- relational Markov networks (RMNs) [Taskar et al 2002]
- Markov logic networks (MLNs) [Richardson & Domingos 2006]
- probabilistic soft logic (PSL) [Broecheler et al 2010]
- FACTORIE [McCallum et al 2009]
- Bayesian logic programs (BLPs) [Kersting & De Raedt 2001]
- relational Bayesian networks (RBNs) [Jaeger 2002]
- logical Bayesian networks (LBNs) [Fierens et al 2005]
- probabilistic relational models (PRMs) [Koller & Pfeffer 1998]
- Bayesian logic (BLOG) [Milch et al 2005]
- CLP(BN) [Santos Costa et al 2008]
- probabilistic programming languages such as ProbLog, PRISM, Church, ...
- and many more ...

Lots of proposals in the literature, e.g.

- relational Markov networks (RMNs) [Taskar et al 2002]
- Markov logic networks (MLNs) [Richardson & Domingos 2006]
- probabilistic soft logic (PSL) [Broecheler et al 2010]
- FACTORIE [McCallum et al 2009]
- Bayesian networks
- relational Bayesian networks
- logic programming
- probabilistic relational models (PRMs) [Koller & Pfeffer 1998]
- Bayesian logic (BLOG) [Milch et al 2005]
- CLP(BN) [Santos Costa et al 2008]
- probabilistic programming languages such as ProbLog, PRISM, Church, ...
- and many more ...

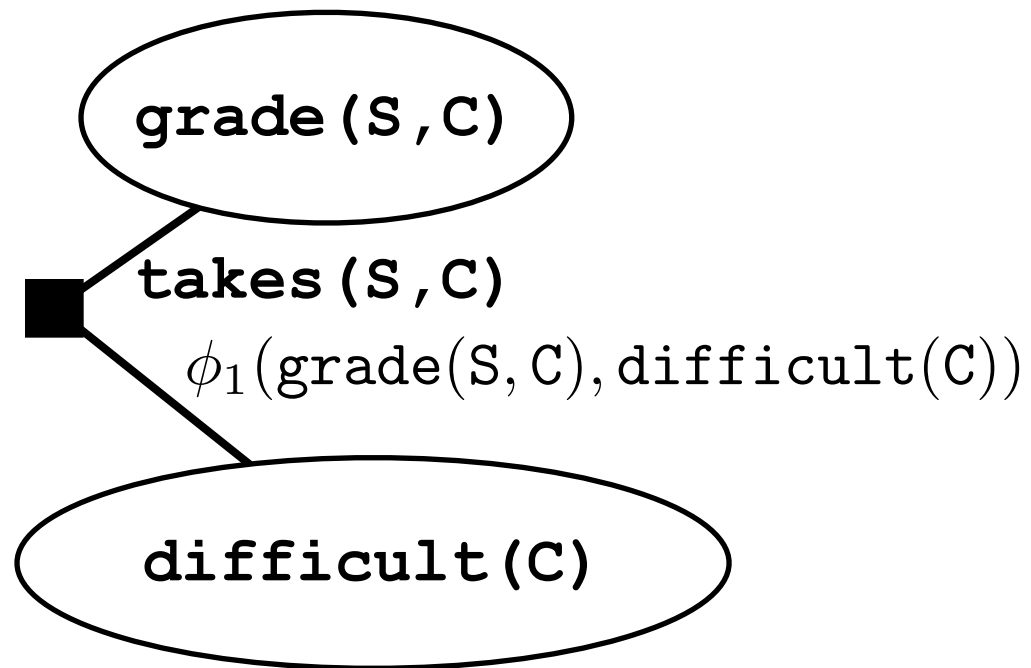
**common principle:
parameterized factor graph
(par-factor graph)**

Par-Factor Graph

[Poole 03]

Par-Factor Graph

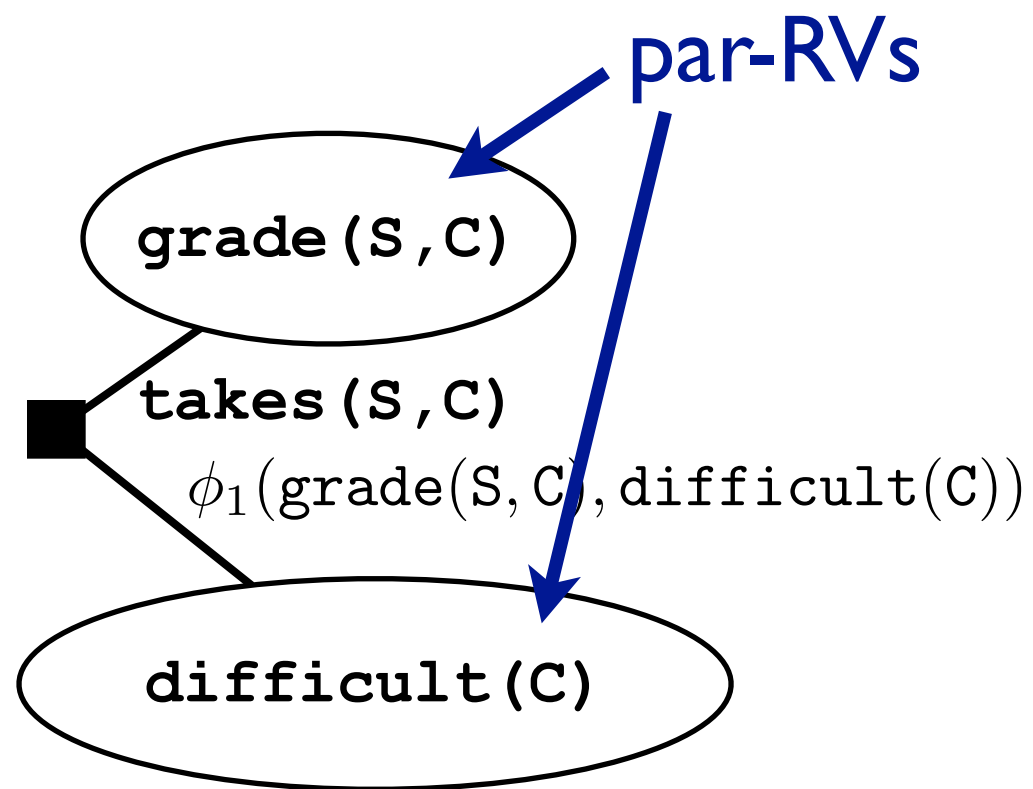
[Poole 03]



parameterized
factor (par-factor)

Par-Factor Graph

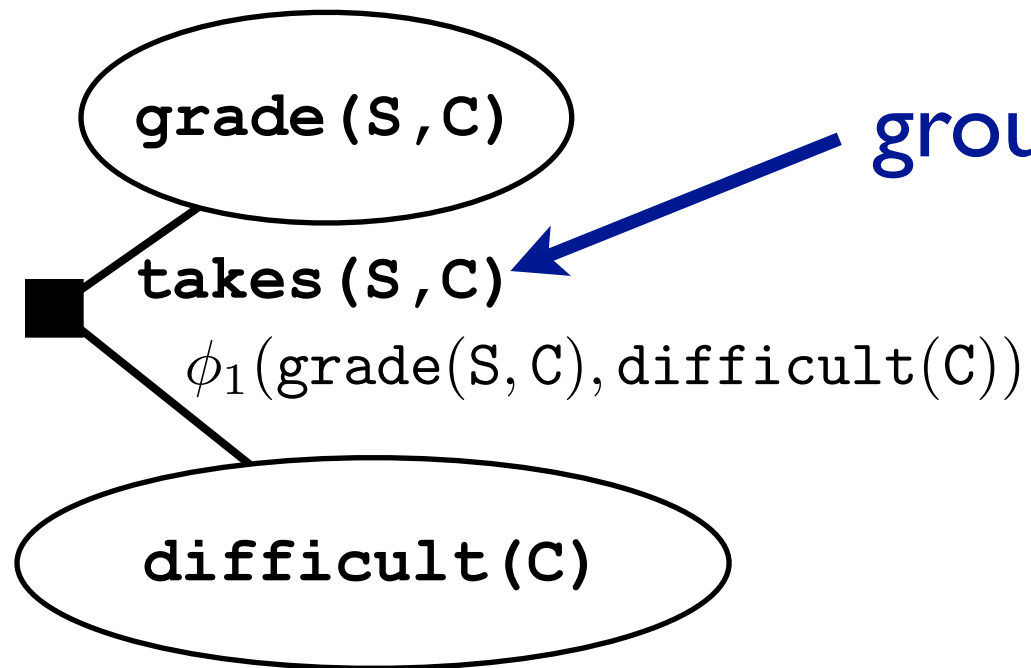
[Poole 03]



parameterized
factor (par-factor)

Par-Factor Graph

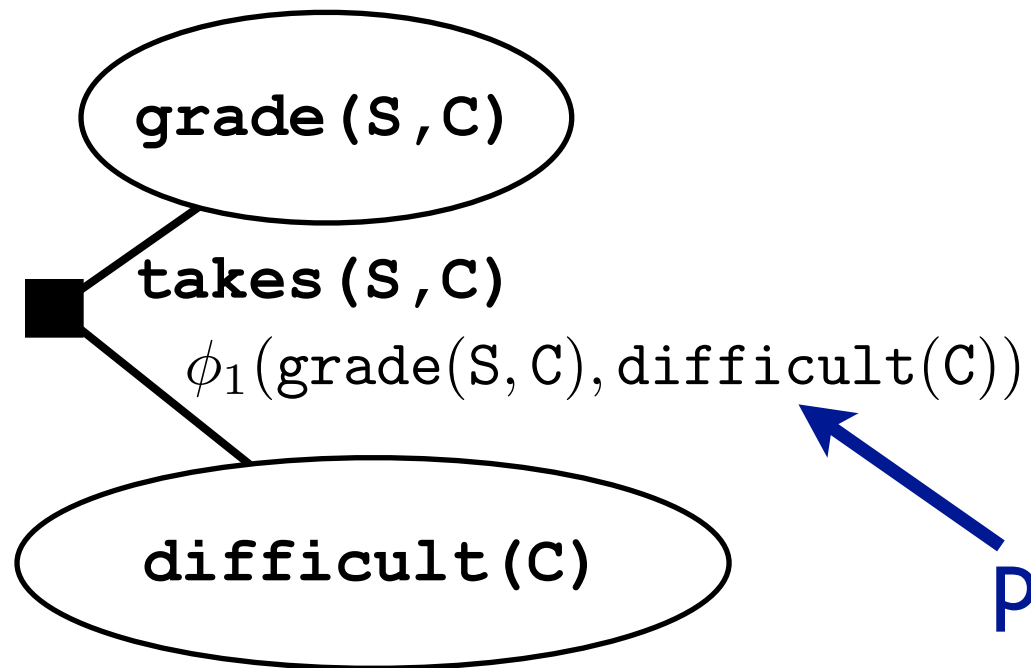
[Poole 03]



**parameterized
factor (par-factor)**

Par-Factor Graph

[Poole 03]

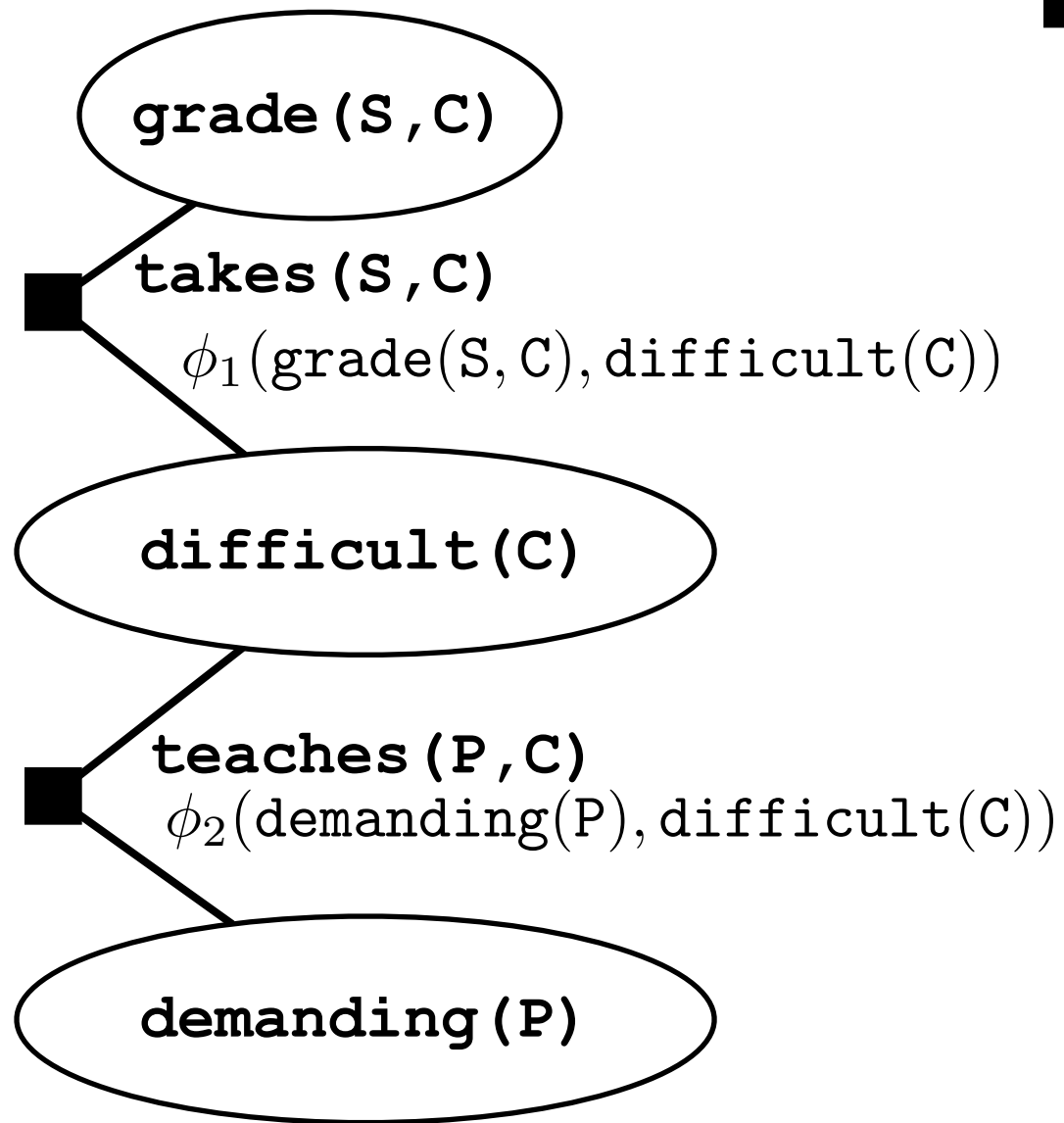


**parameterized
factor (par-factor)**

potential function

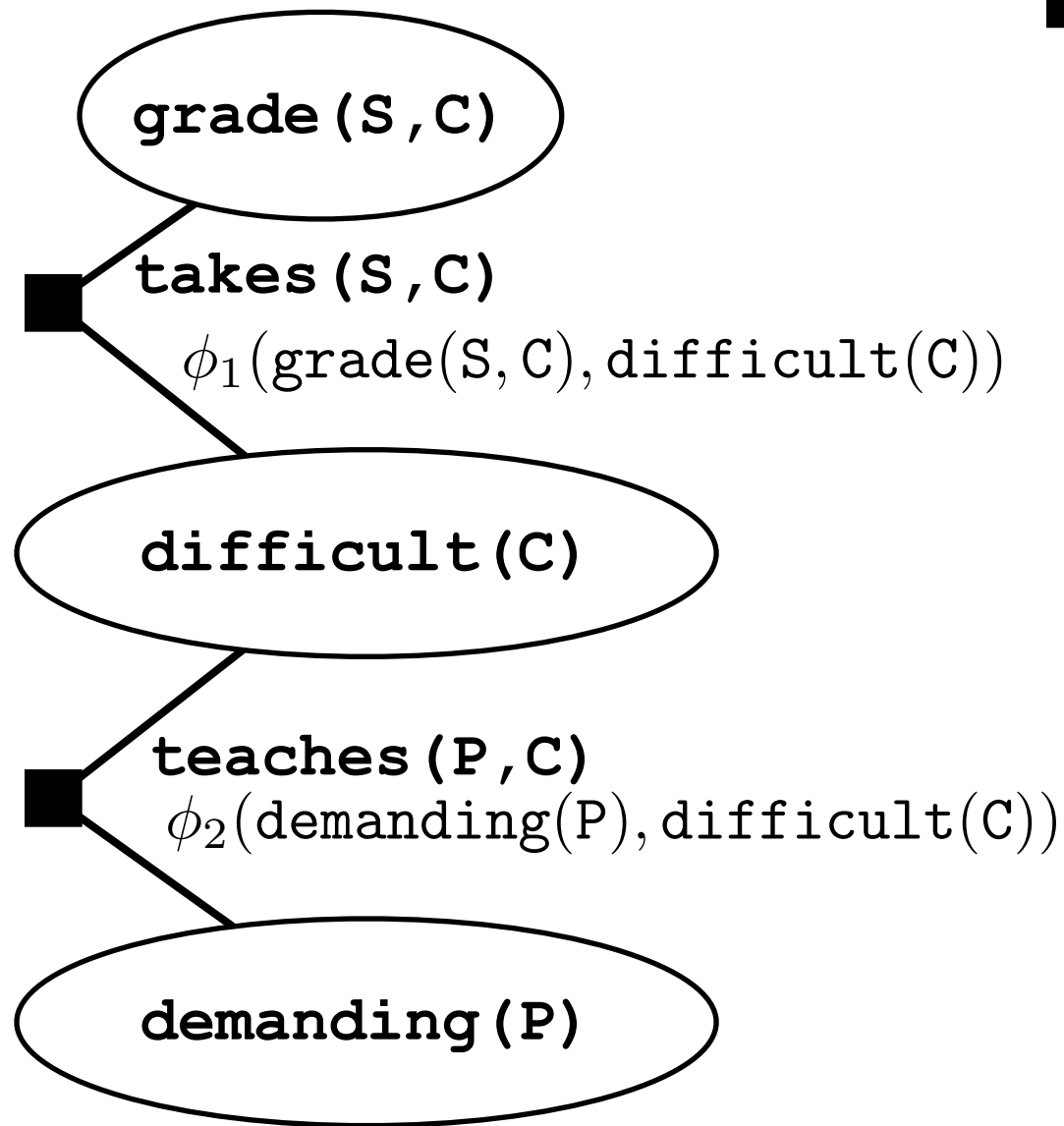
Par-Factor Graph

[Poole 03]



Par-Factor Graph

[Poole 03]

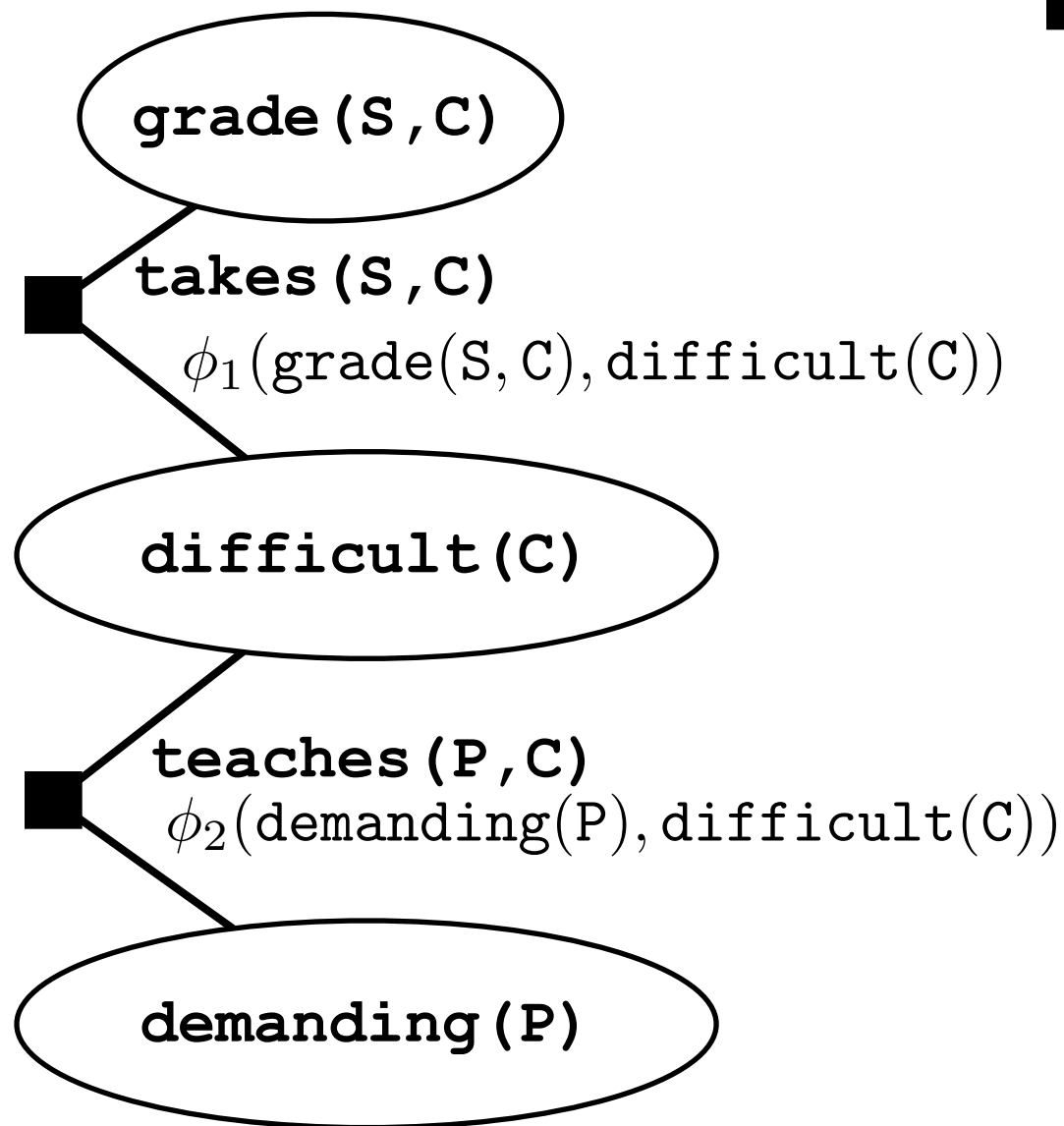


$$P(.) = \frac{1}{Z} \cdot \prod_{\text{takes}(s,c)} \phi_1(\text{grade}(s, c), \text{difficult}(c))$$

$$\cdot \prod_{\text{teaches}(p,c)} \phi_2(\text{demanding}(p), \text{difficult}(c))$$

Par-Factor Graph

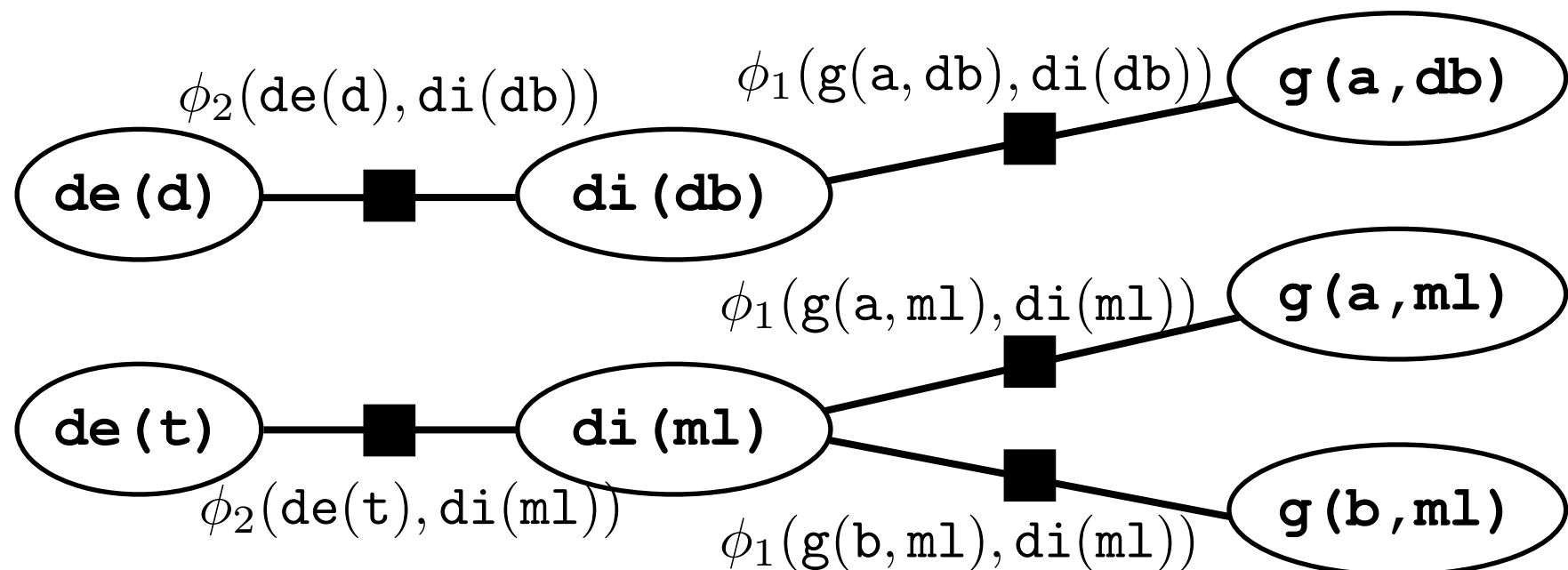
[Poole 03]



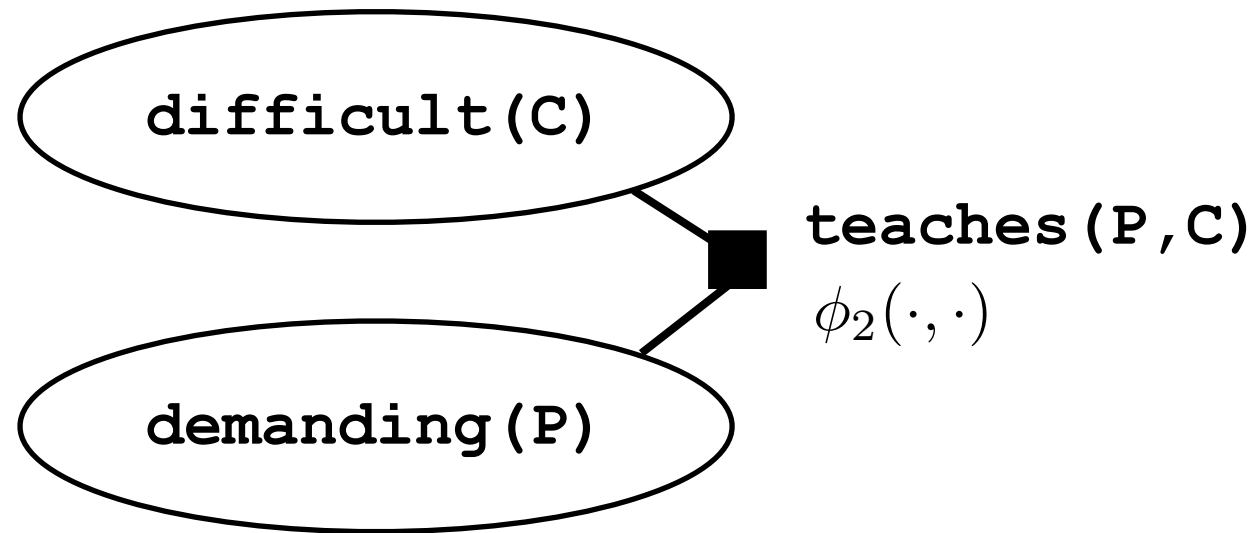
$$P(.) = \frac{1}{Z} \cdot \prod_{\text{takes}(s,c)} \phi_1(\text{grade}(s, c), \text{difficult}(c))$$

$$\cdot \prod_{\text{teaches}(p,c)} \phi_2(\text{demanding}(p), \text{difficult}(c))$$

takes (ann, ml) .
 takes (bob, ml) .
 takes (ann, db) .
 teaches (dan, db) .
 teaches (tom, ml) .

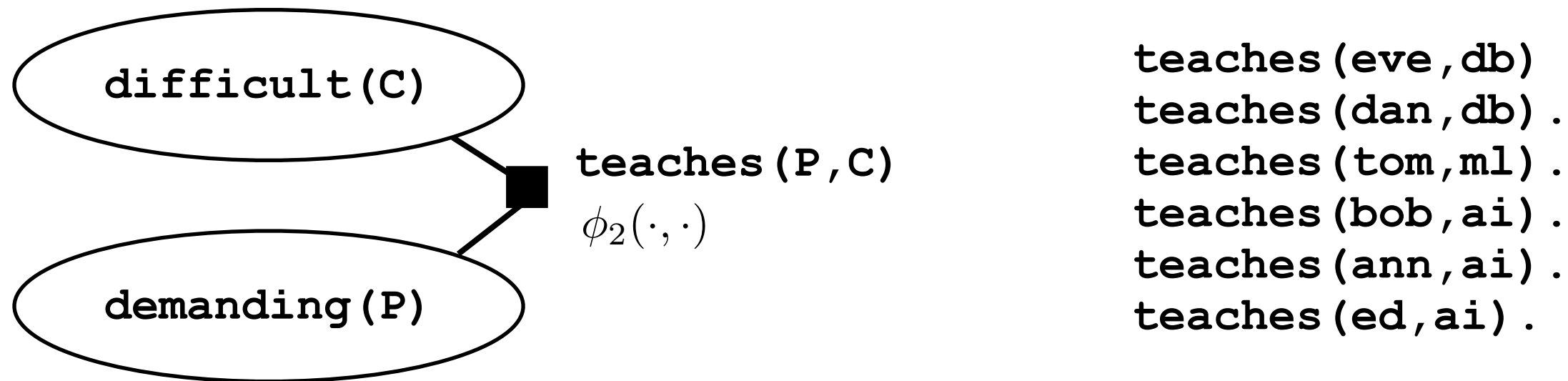


What if multiple professors teach the same course?



```
teaches(eve,db)
teaches(dan,db).
teaches(tom,ml).
teaches(bob,ai).
teaches(ann,ai).
teaches(ed,ai).
```

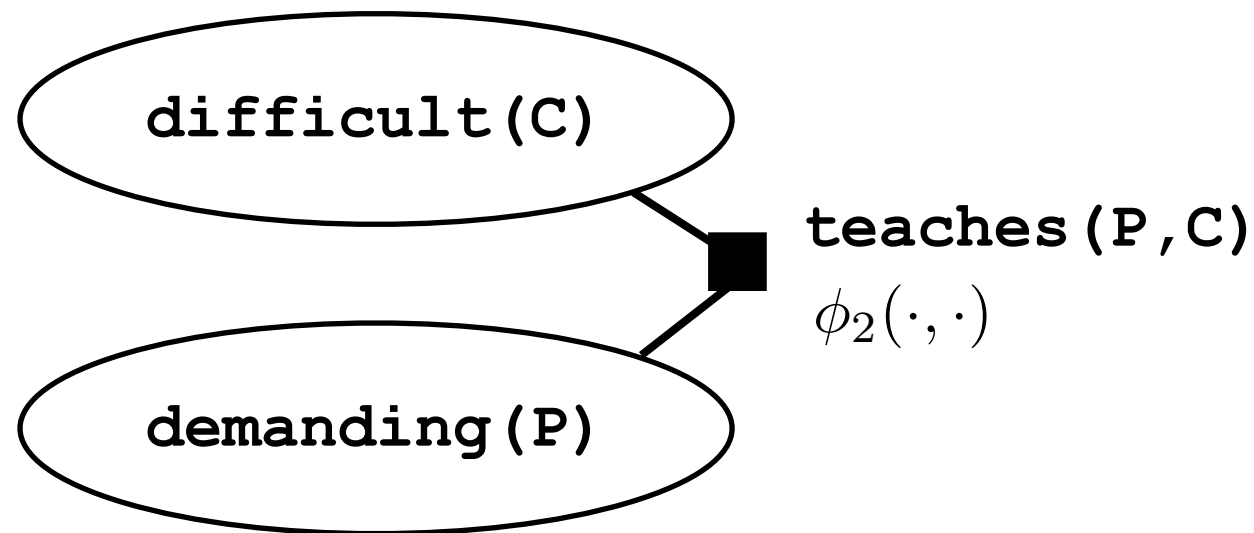
What if multiple professors teach the same course?



Option 1: aggregate values of RVs, then compute potential

$$\phi_2(\text{demanding}(P), \text{difficult}(C)) = P(\text{difficult}(C) | \text{mean}\{\text{demanding}(P)\})$$

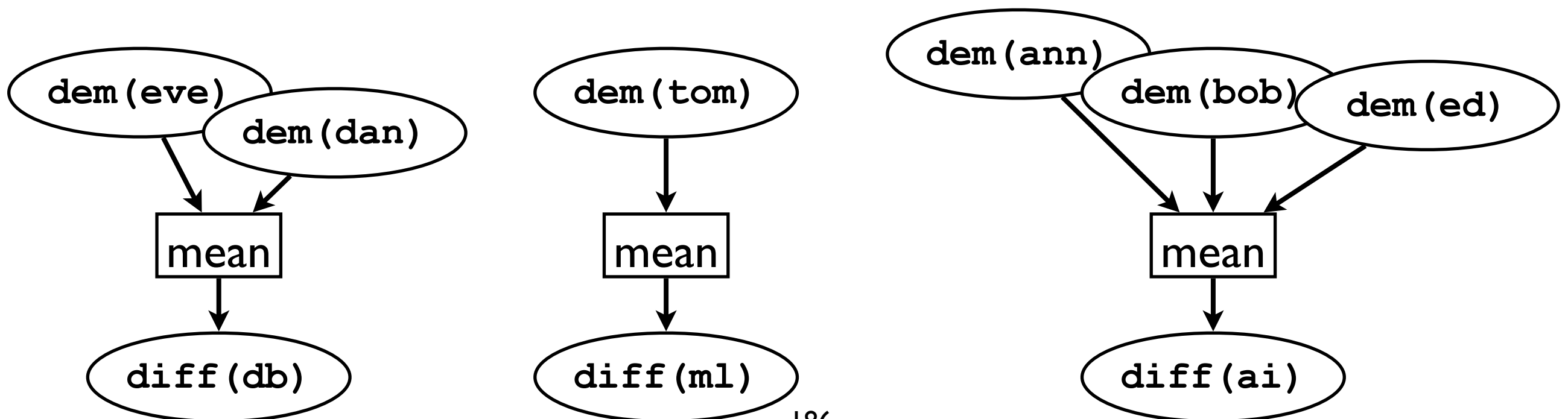
What if multiple professors teach the same course?



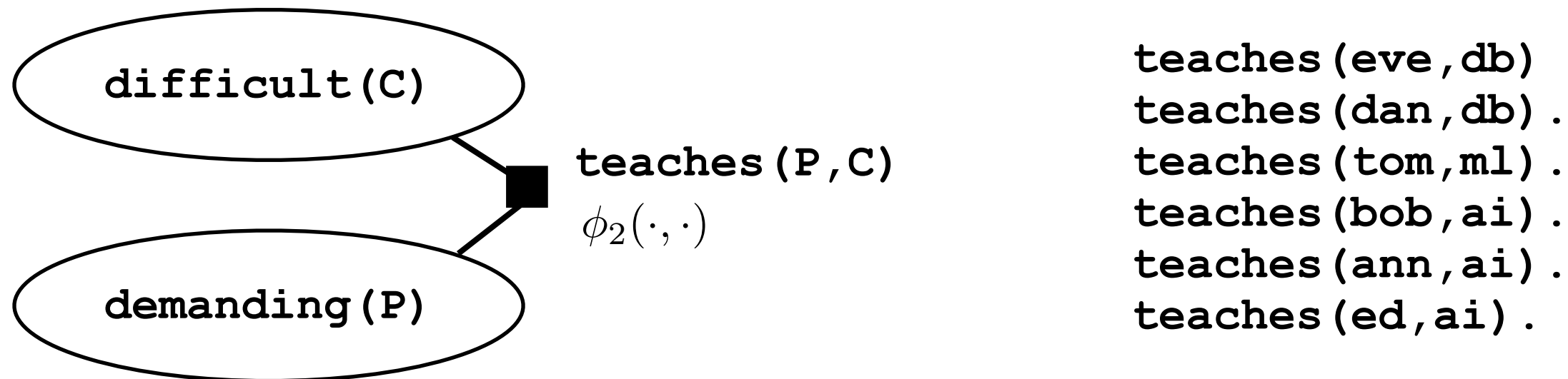
```
teaches(eve,db)
teaches(dan,db).
teaches(tom,ml).
teaches(bob,ai).
teaches(ann,ai).
teaches(ed,ai).
```

Option 1: aggregate values of RVs, then compute potential

$$\phi_2(\text{demanding}(P), \text{difficult}(C)) = P(\text{difficult}(C) | \text{mean}\{\text{demanding}(P)\})$$



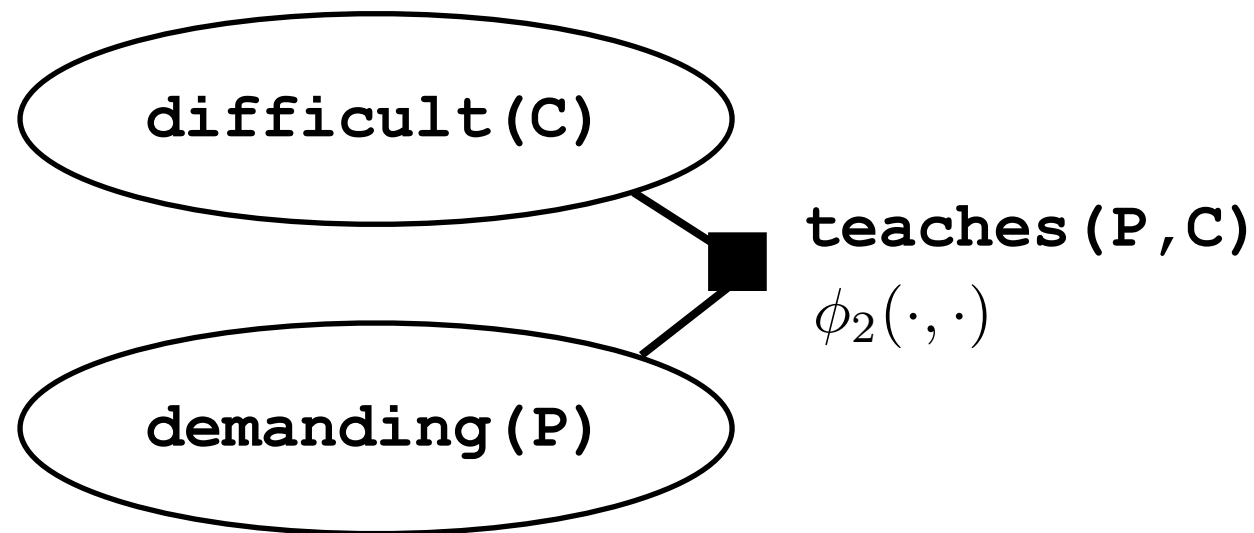
What if multiple professors teach the same course?



Option 2: compute potential for each RV, then combine

$$\phi_2(\text{demanding}(P), \text{difficult}(C)) = 1 - \prod_P (1 - P(\text{difficult}(C) | \text{demanding}(P)))$$

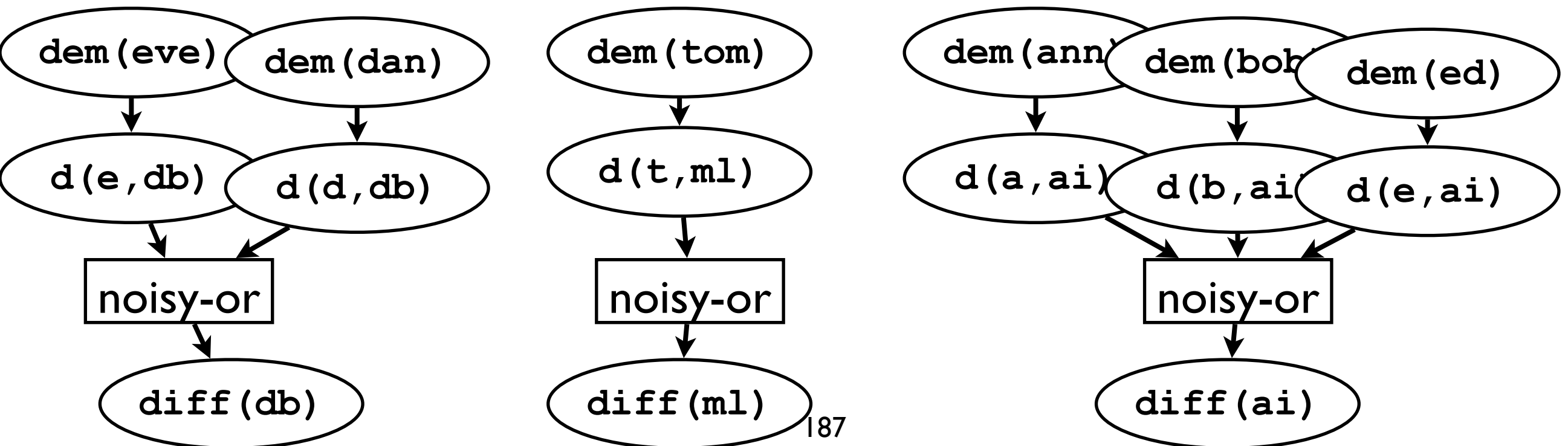
What if multiple professors teach the same course?



```
teaches(eve, db)
teaches(dan, db) .
teaches(tom, ml) .
teaches(bob, ai) .
teaches(ann, ai) .
teaches(ed, ai) .
```

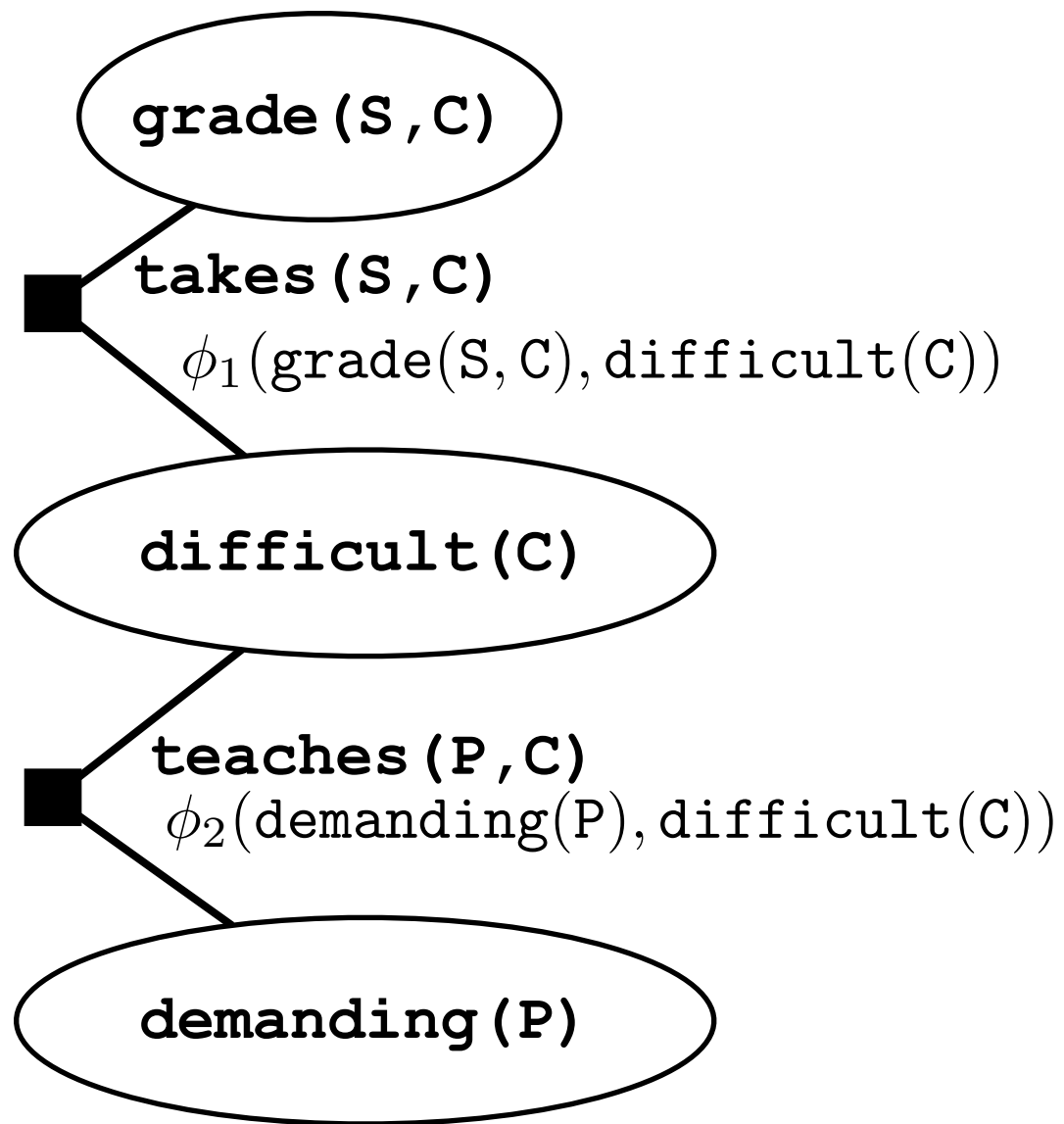
Option 2: compute potential for each RV, then combine

$$\phi_2(\text{demanding}(P), \text{difficult}(C)) = 1 - \prod_P (1 - P(\text{difficult}(C) | \text{demanding}(P)))$$



Inference by Grounding

demanding (tom) ?

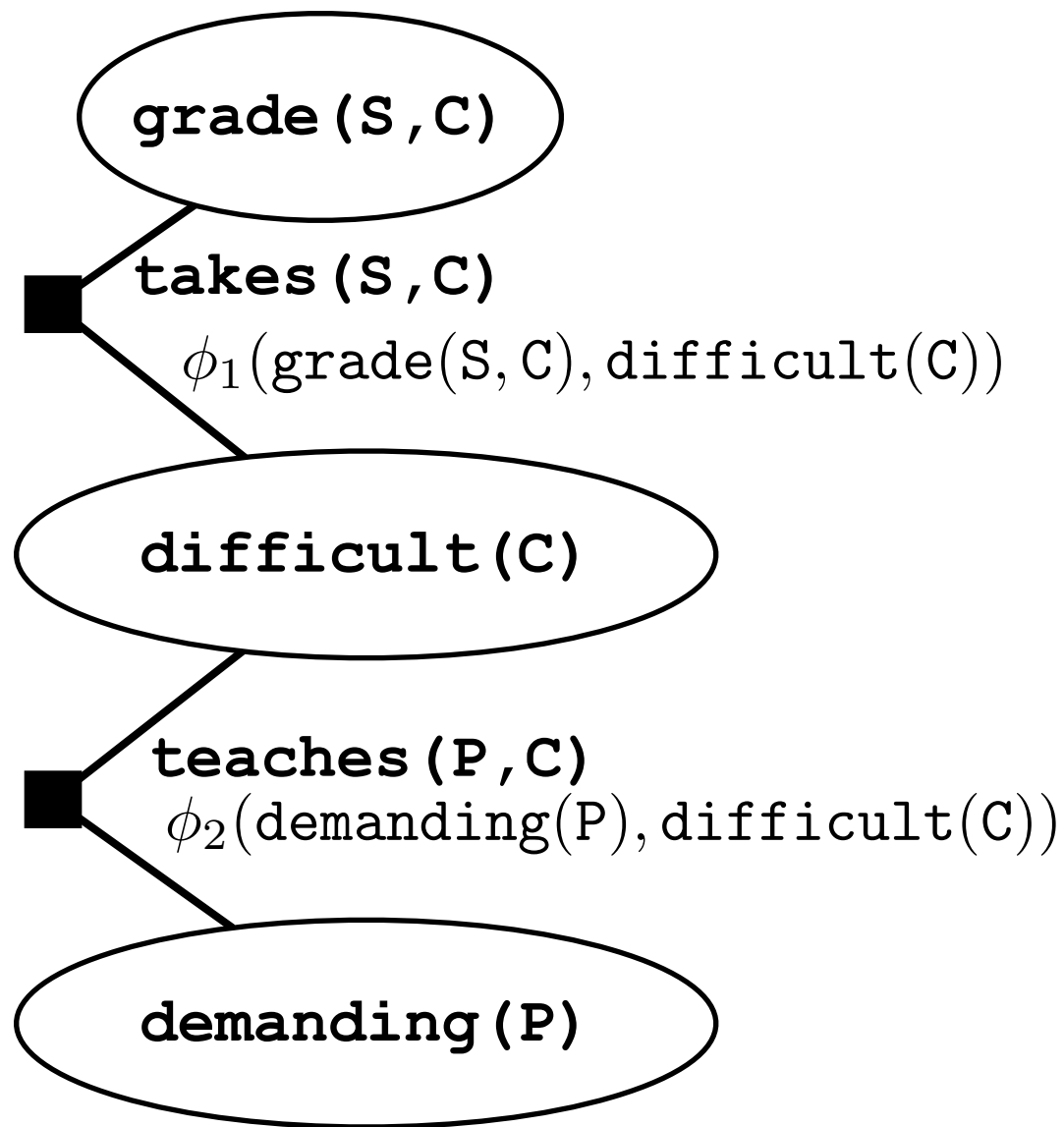


```
takes (ann, ml) .  
takes (bob, ml) .  
takes (ann, db) .  
teaches (dan, db) .  
teaches (tom, ml) .
```

Inference by Grounding

demanding (tom) ?

1. construct factor graph
2. run any propositional inference technique

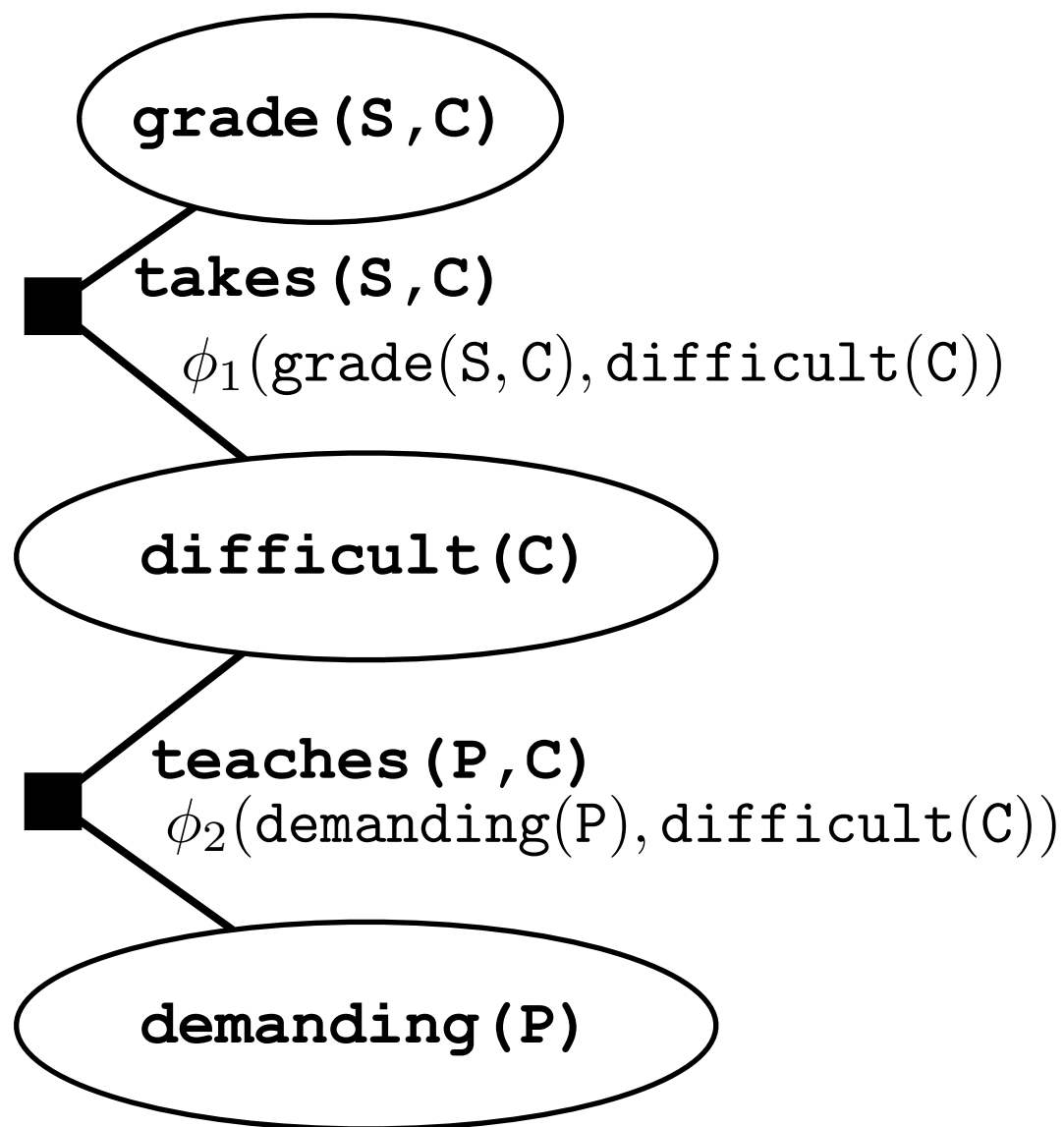


```
takes (ann , ml) .  
takes (bob , ml) .  
takes (ann , db) .  
teaches (dan , db) .  
teaches (tom , ml) .
```

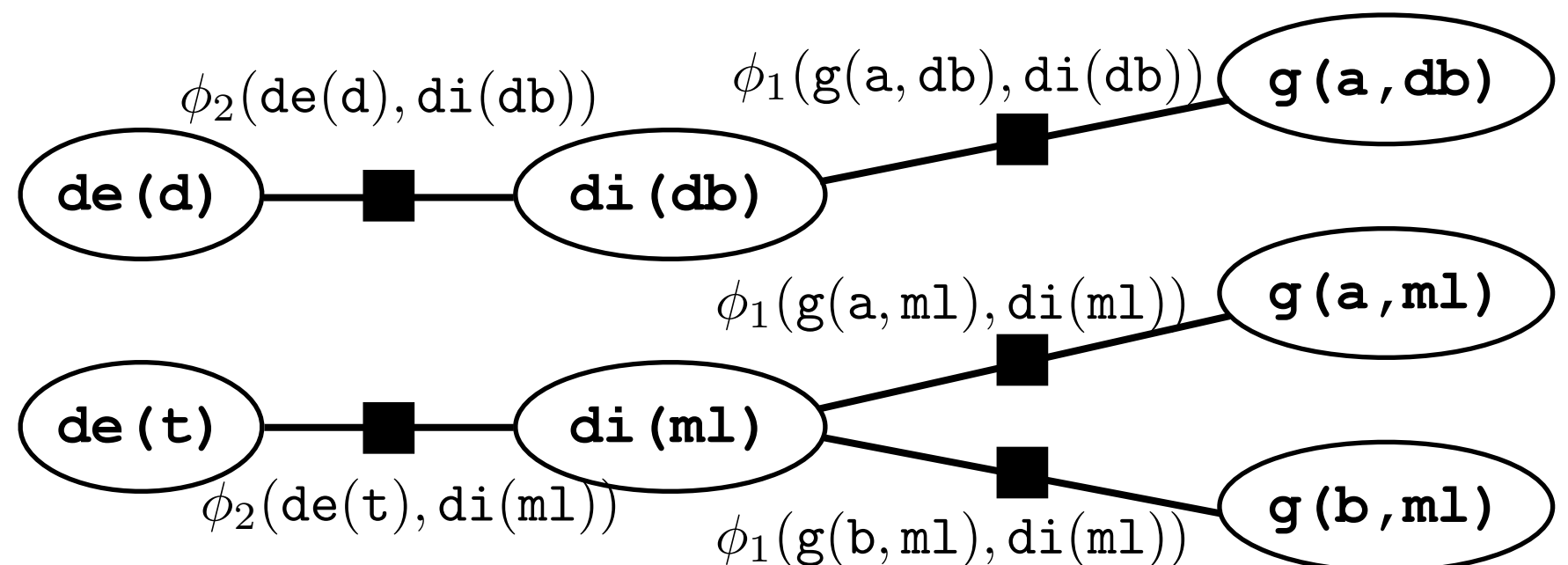
Inference by Grounding

demanding (tom) ?

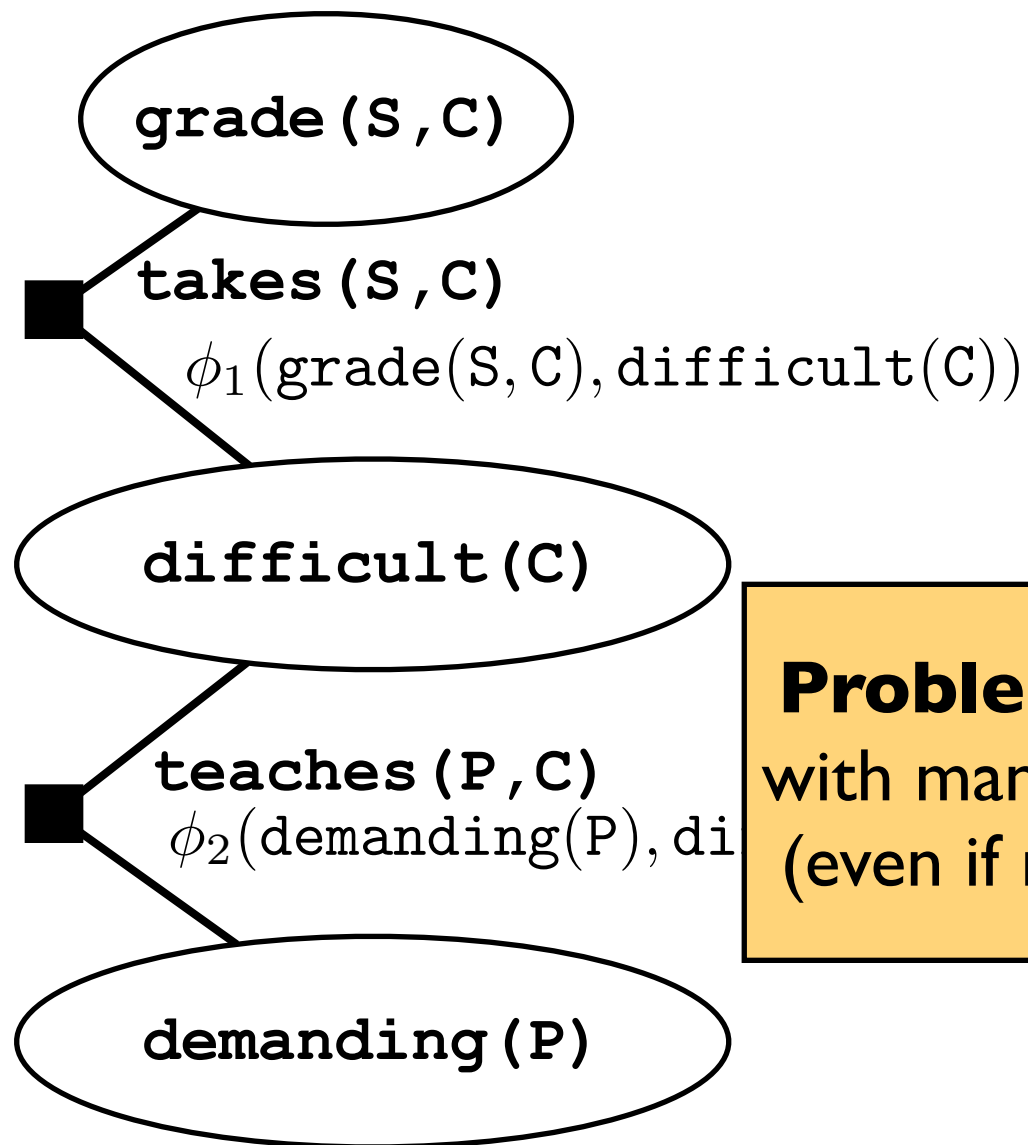
1. construct factor graph
2. run any propositional inference technique



`takes(ann, ml) .`
`takes(bob, ml) .`
`takes(ann, db) .`
`teaches(dan, db) .`
`teaches(tom, ml) .`



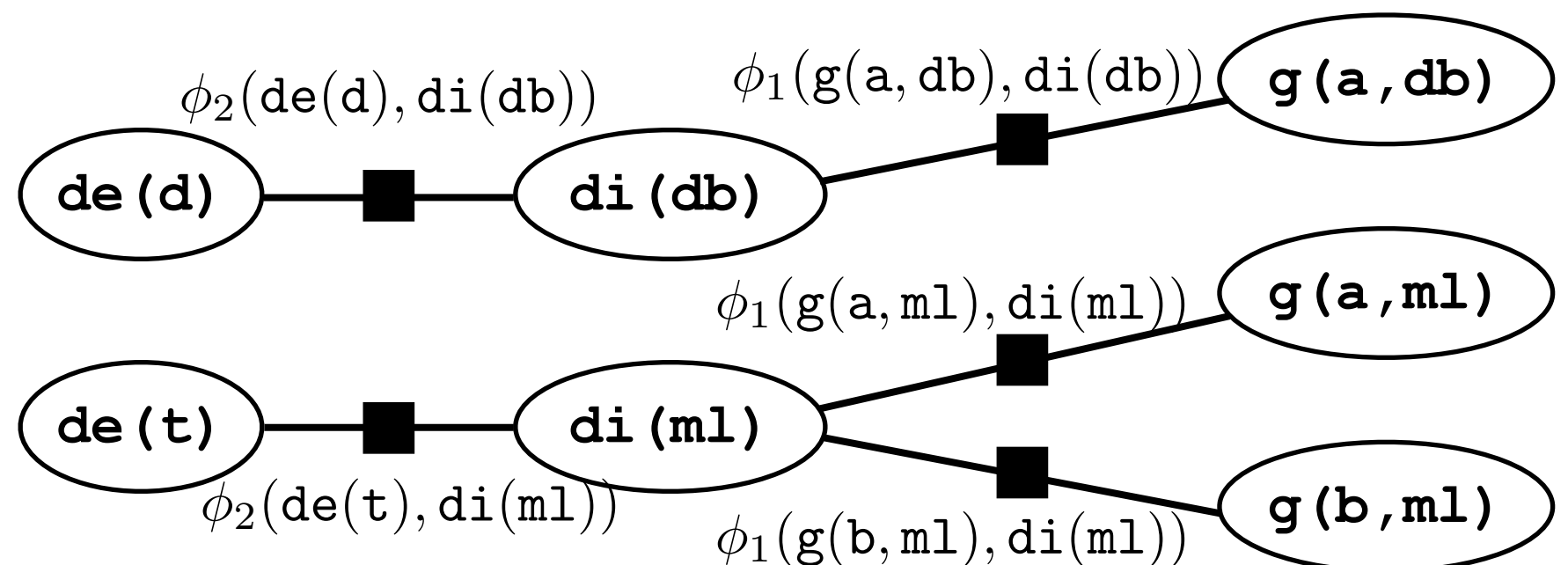
Inference by Grounding

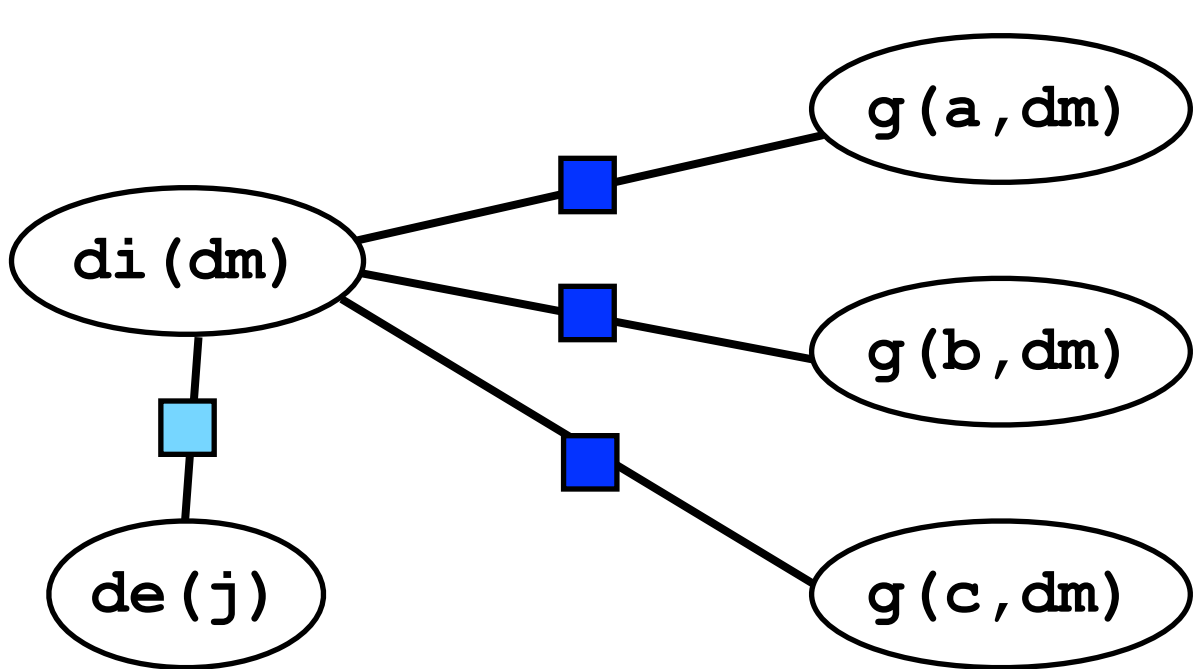
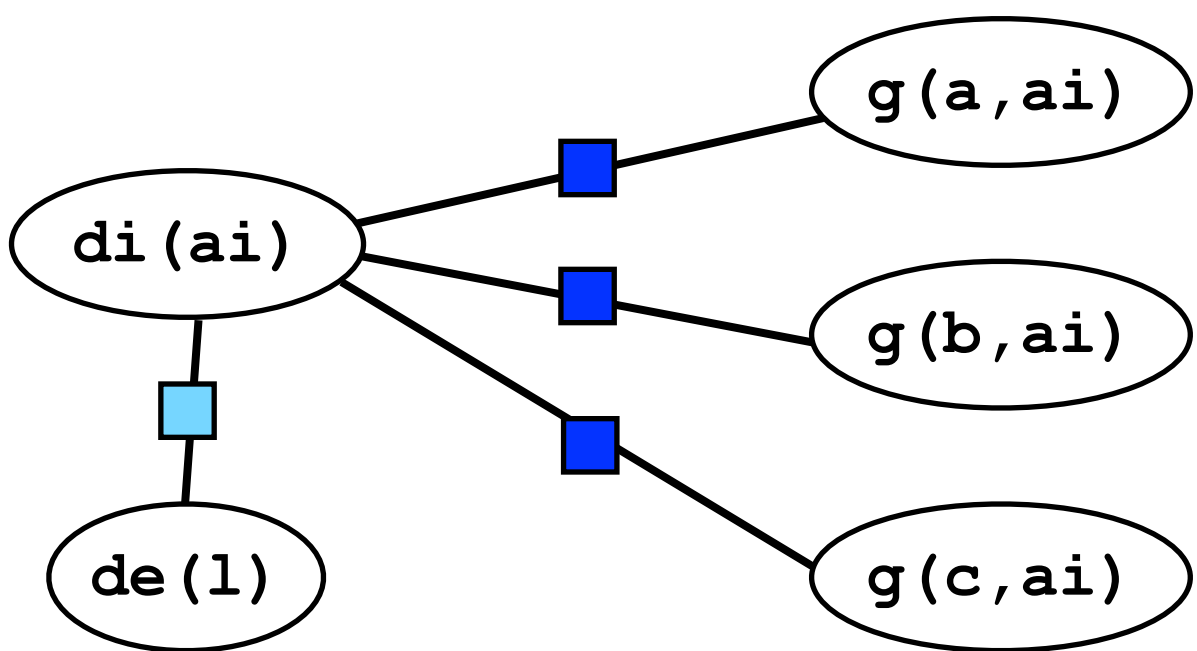
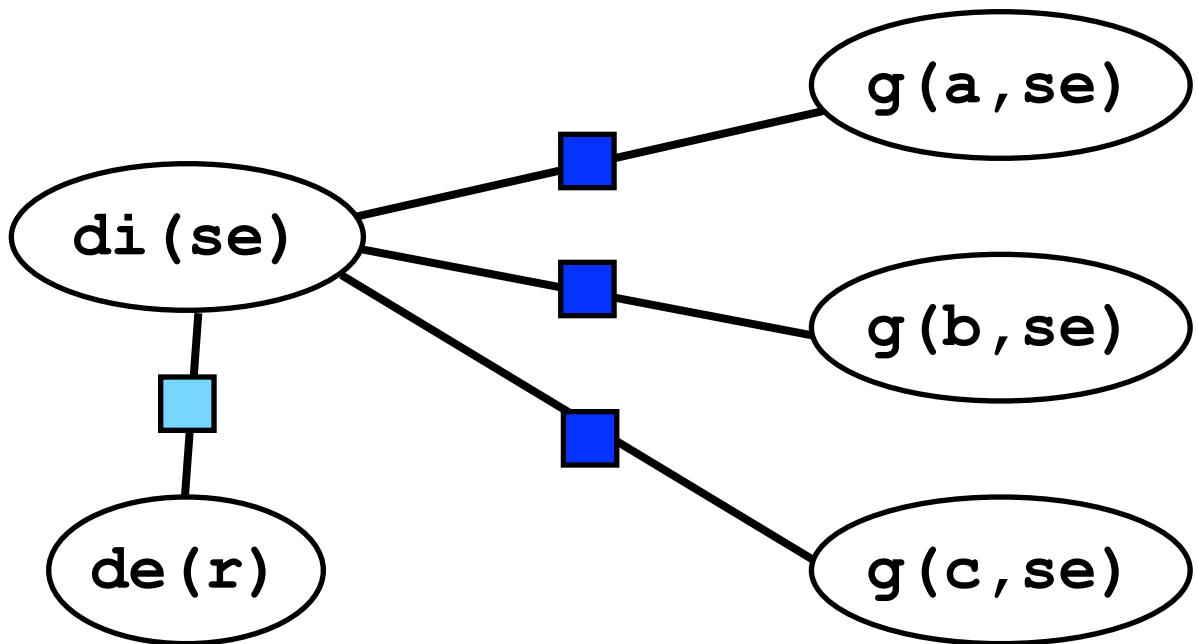
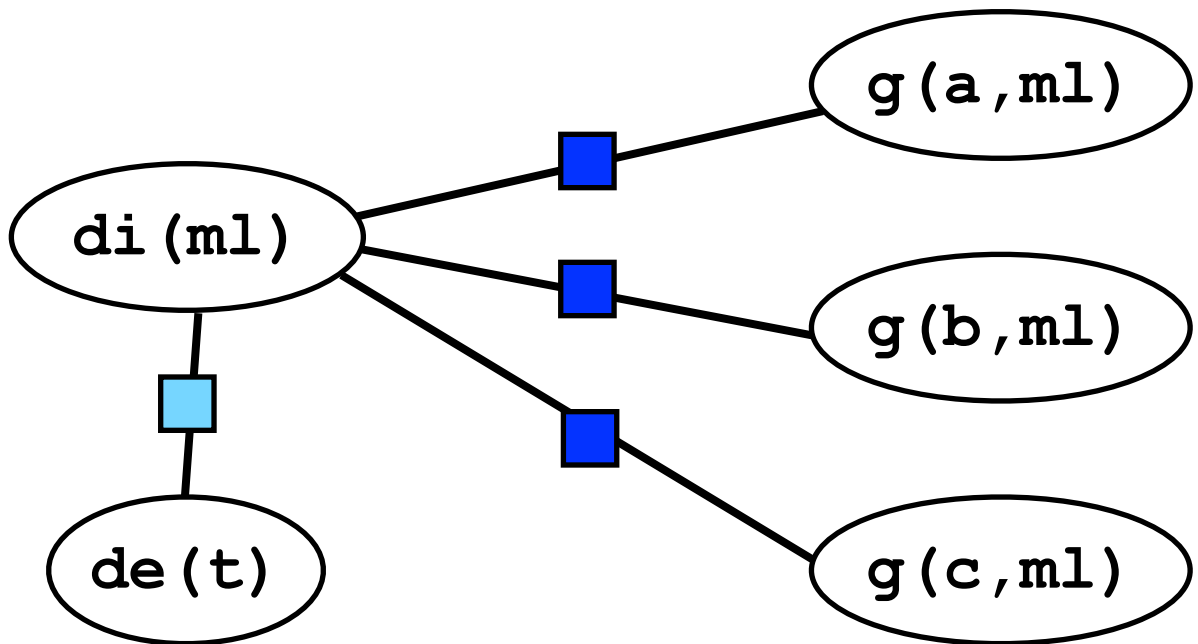
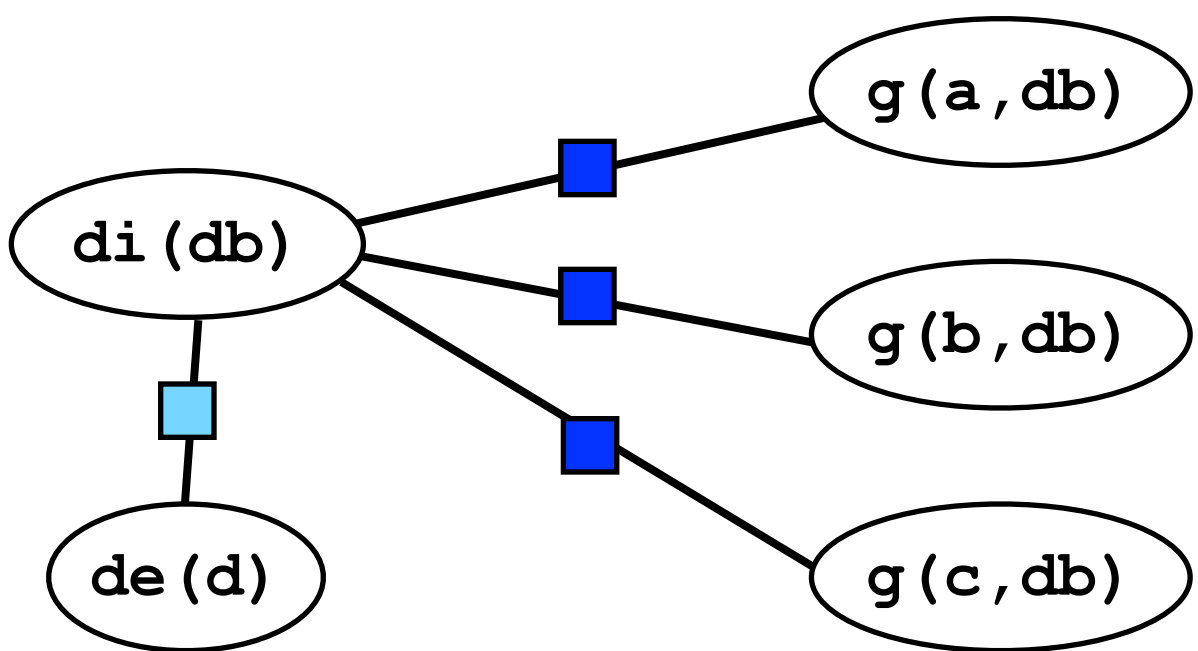


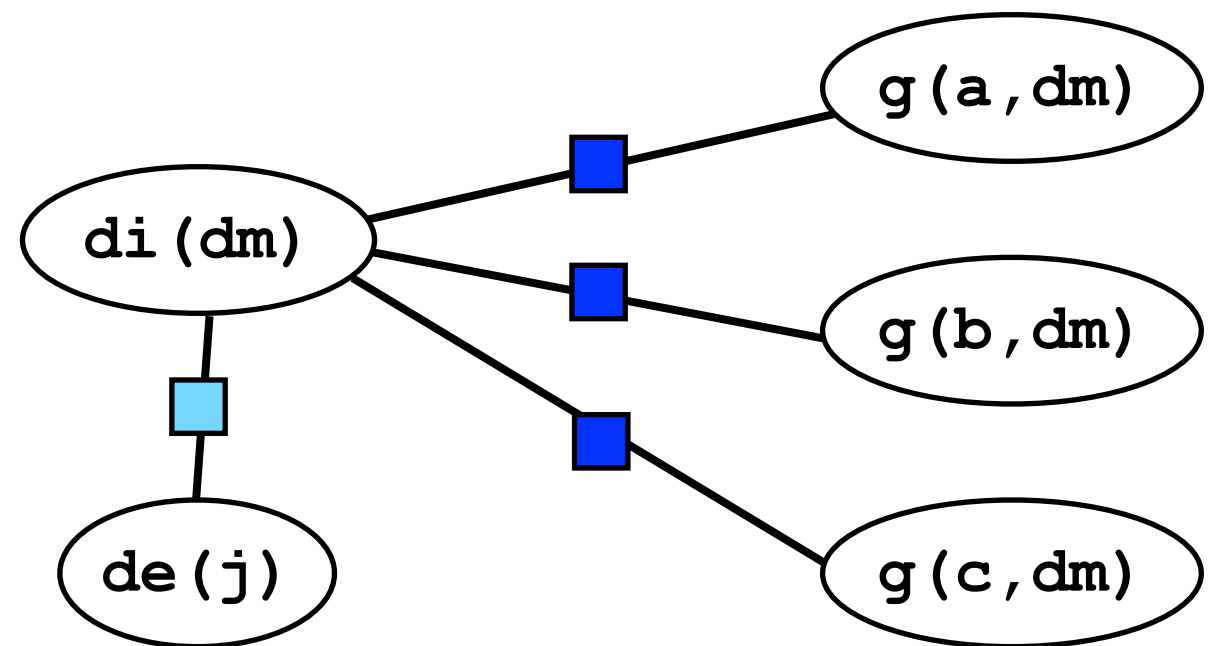
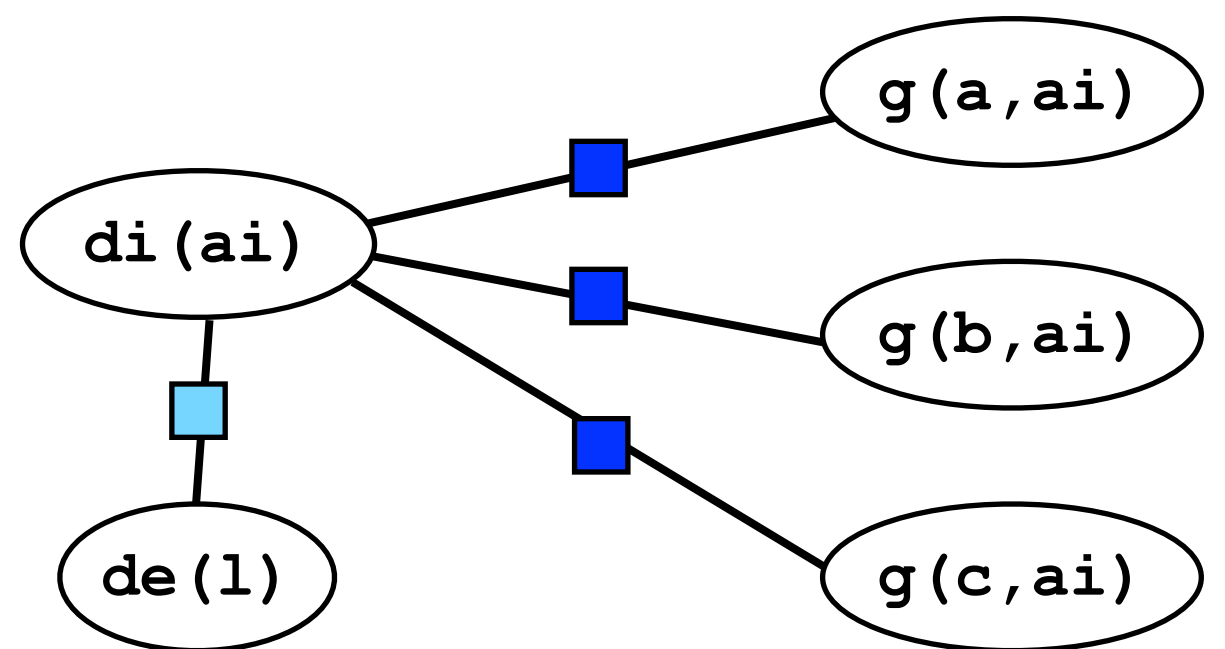
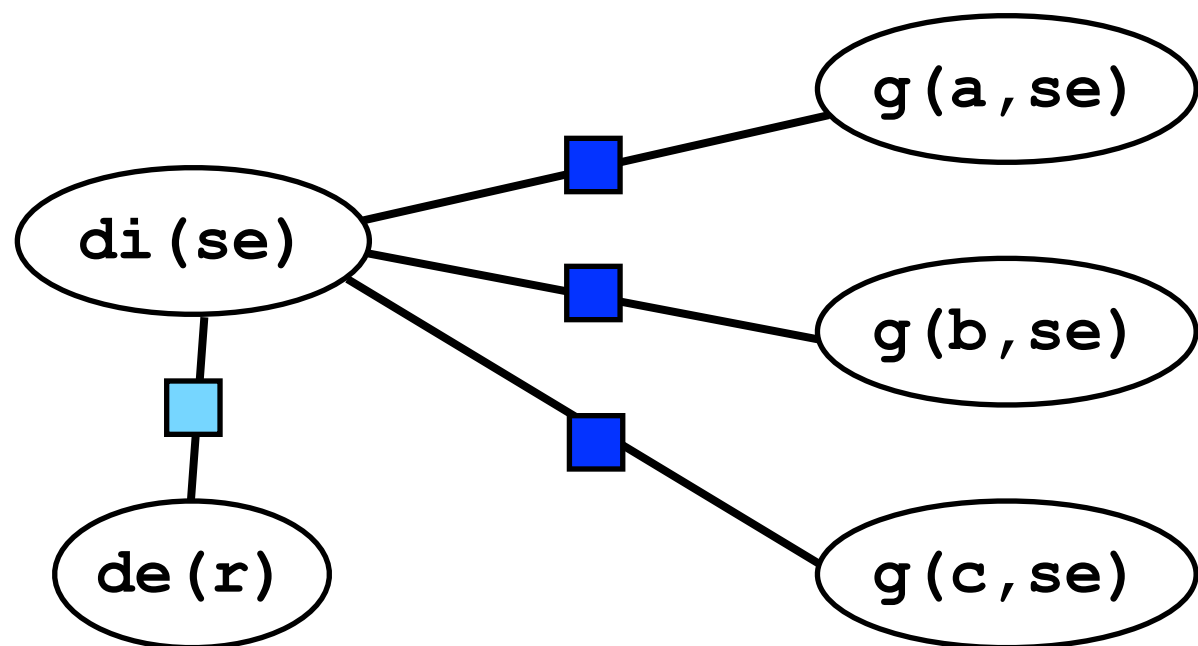
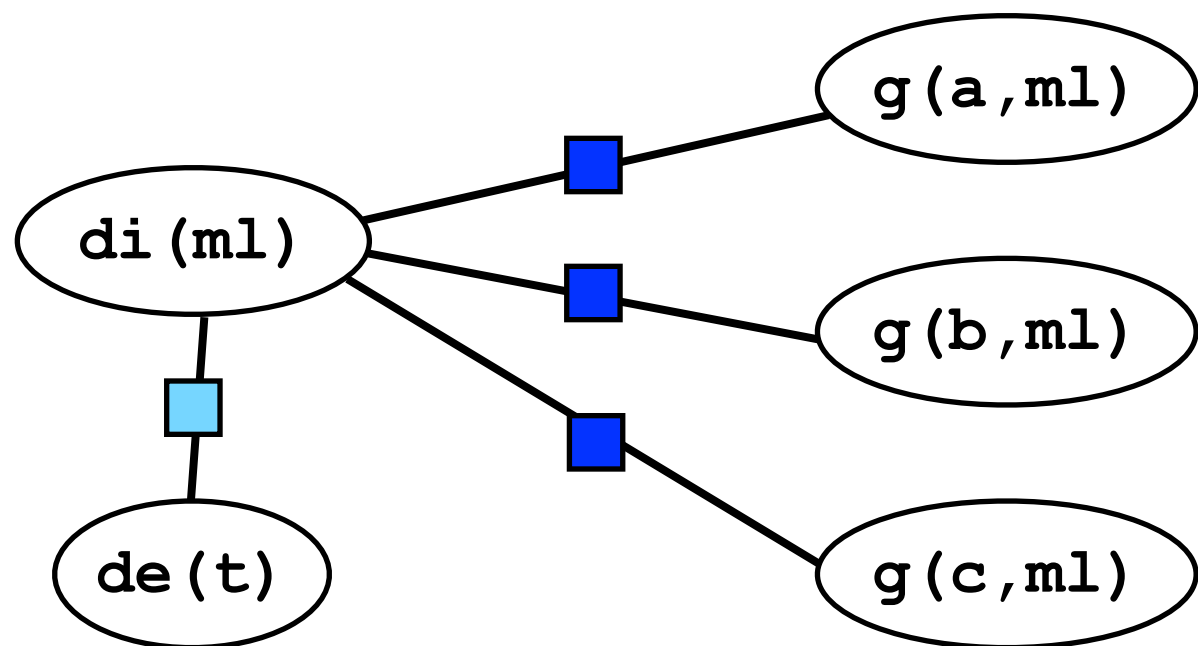
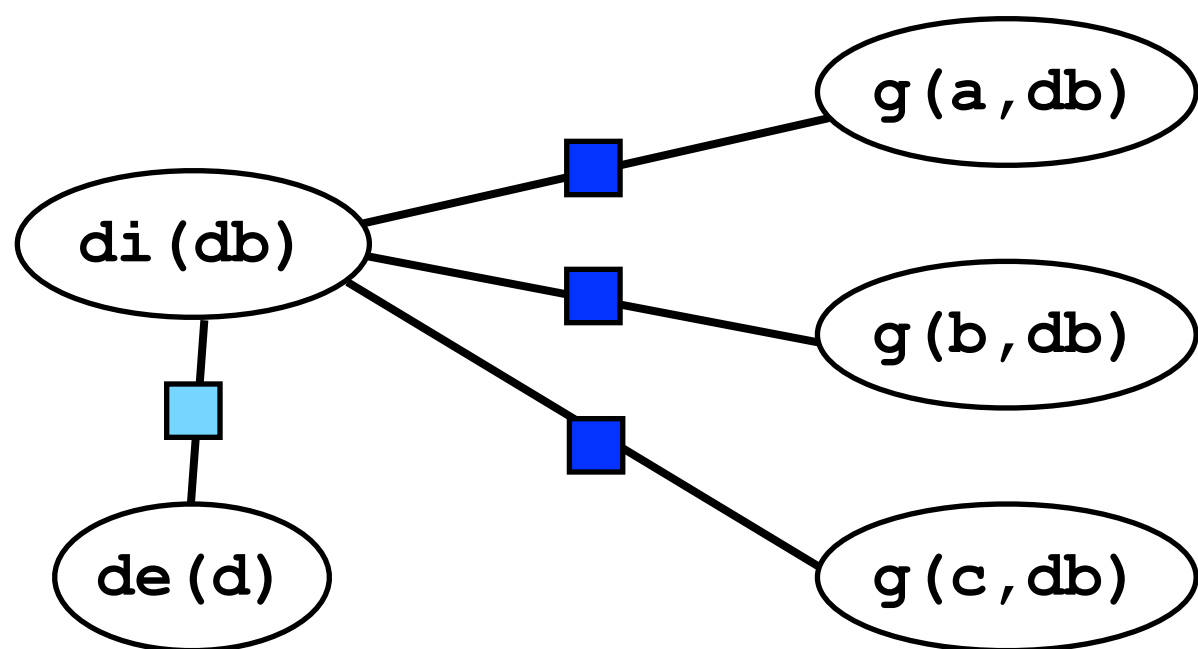
Problem: often huge factor graphs with many repetitions or symmetries (even if restricted to relevant parts)

1. run any propositional inference technique

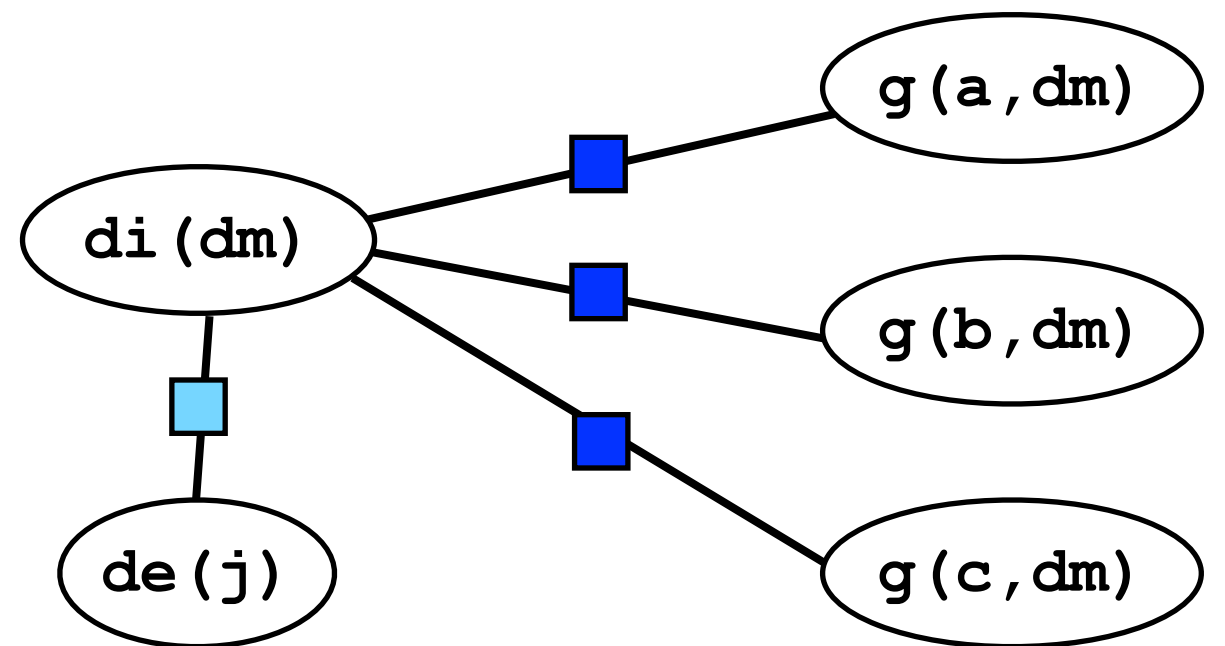
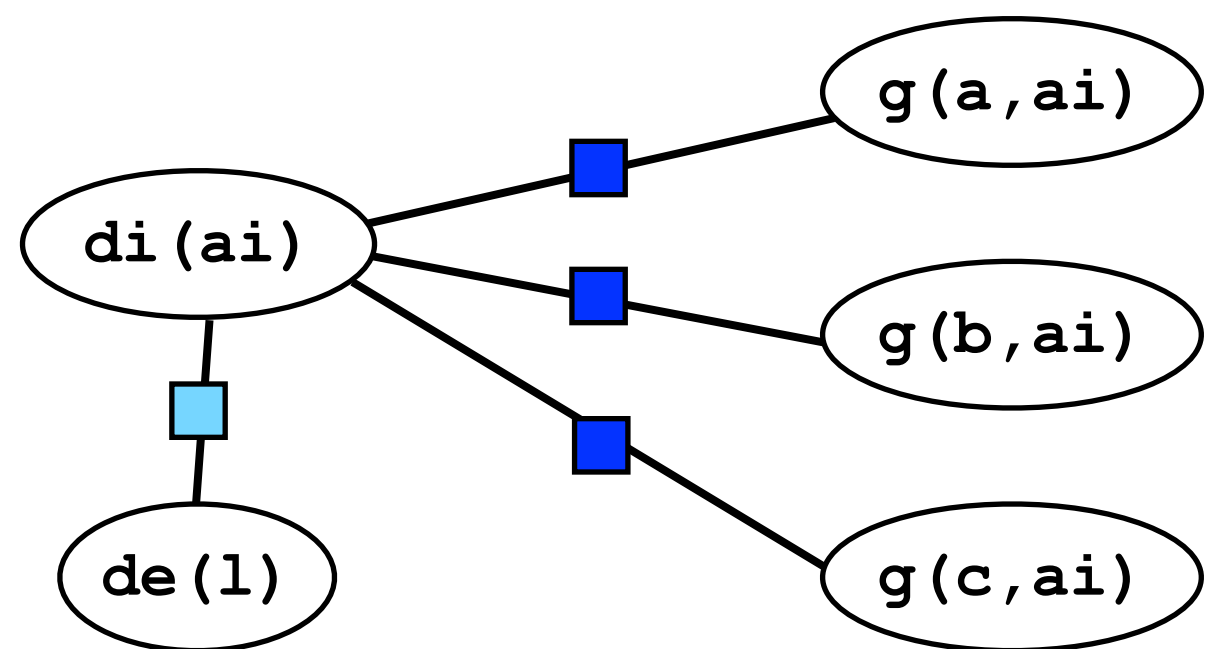
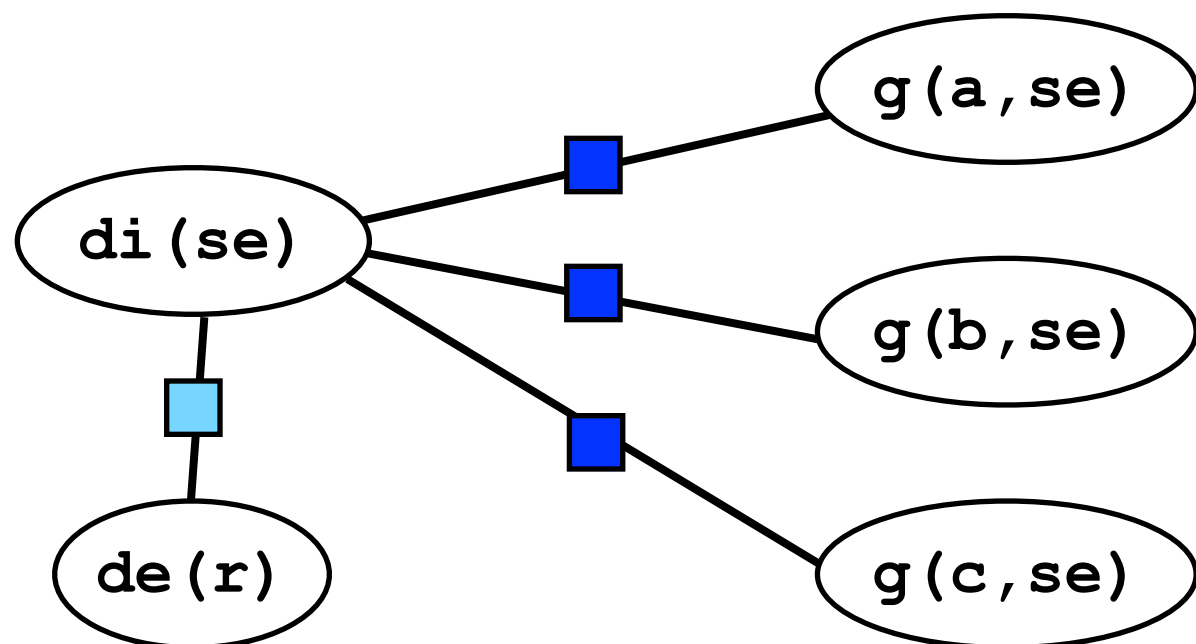
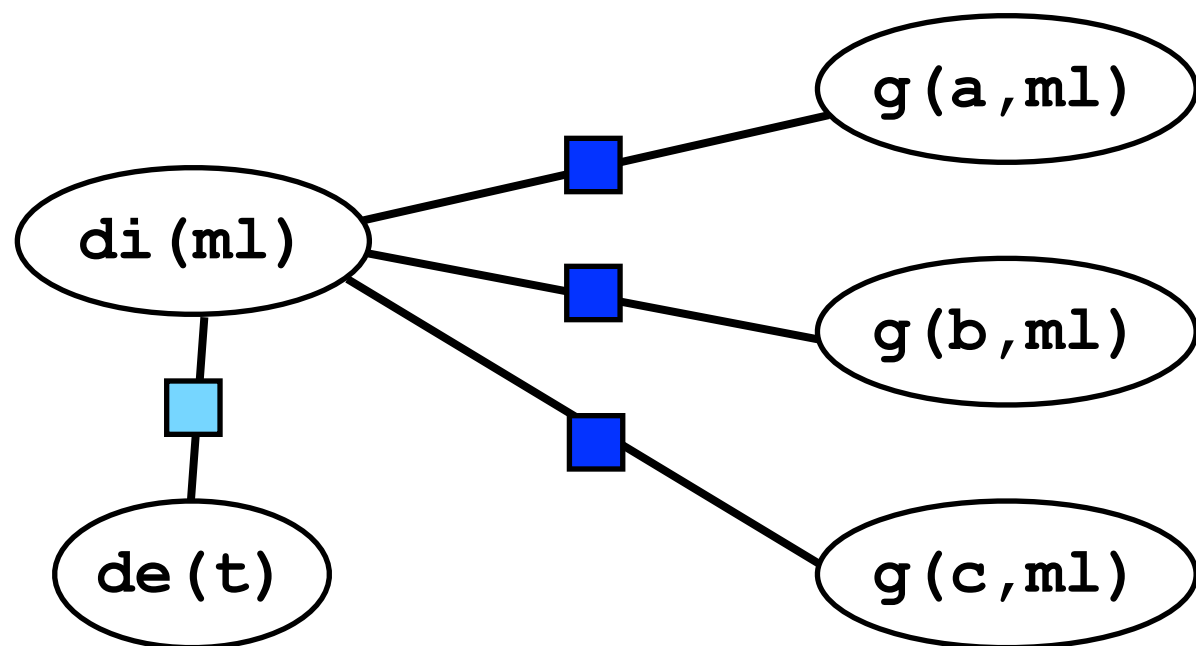
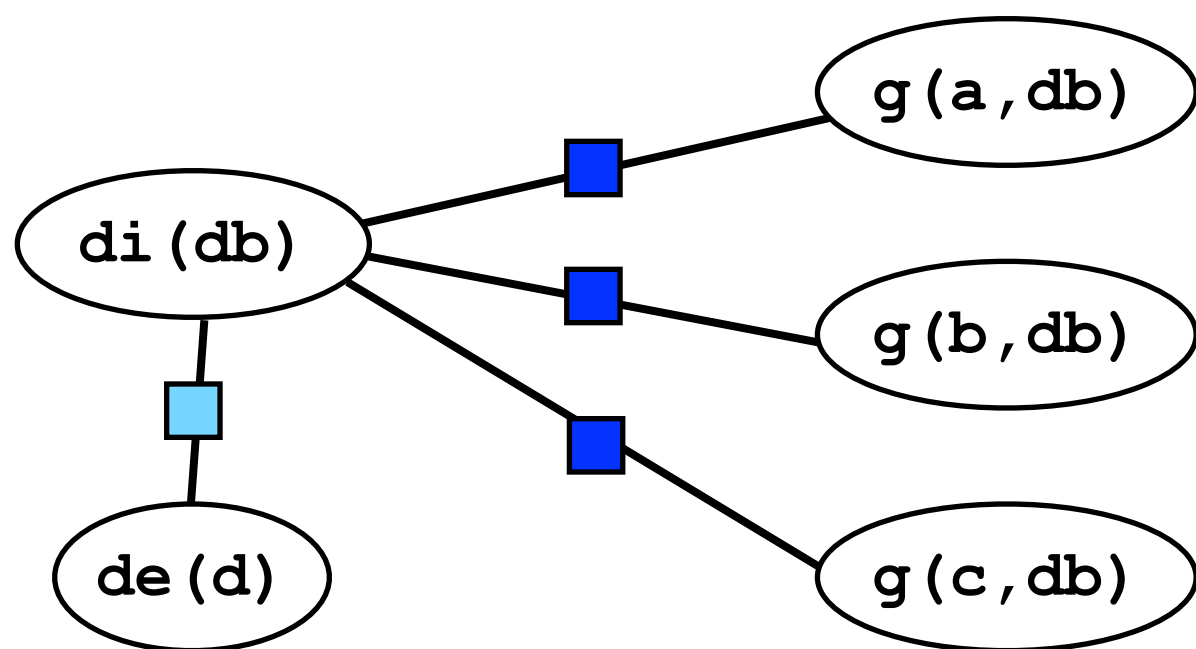
$\text{takes}(\text{ann}, \text{ml}) .$
 $\text{takes}(\text{bob}, \text{ml}) .$
 $\text{takes}(\text{ann}, \text{db}) .$
 $\text{teaches}(\text{dan}, \text{db}) .$
 $\text{teaches}(\text{tom}, \text{ml}) .$



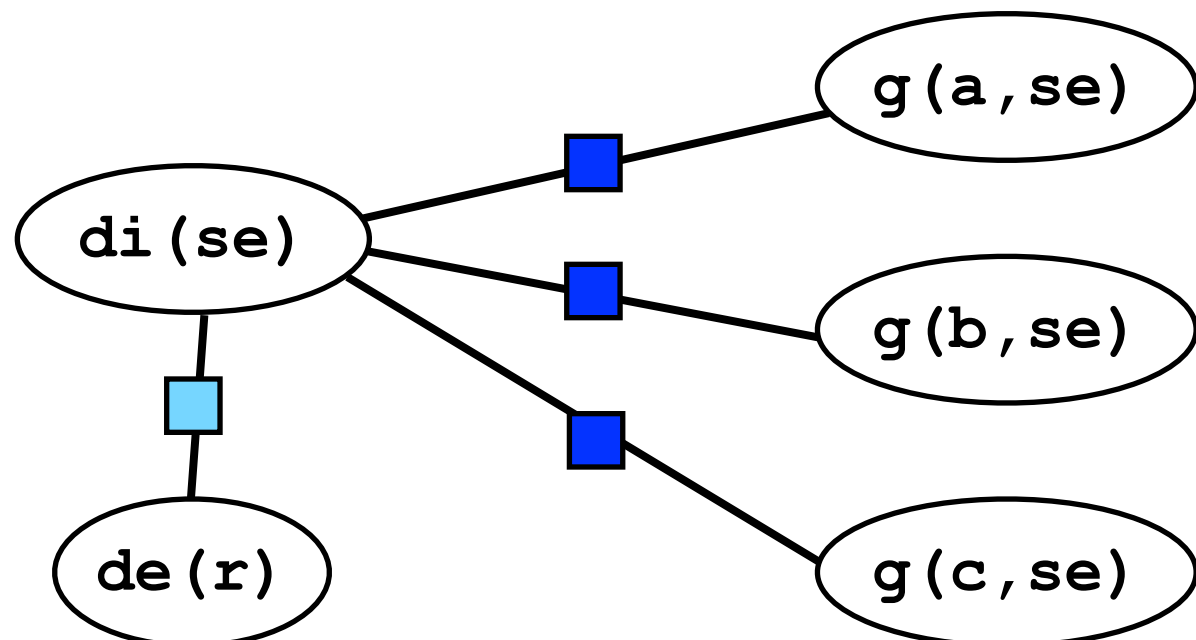
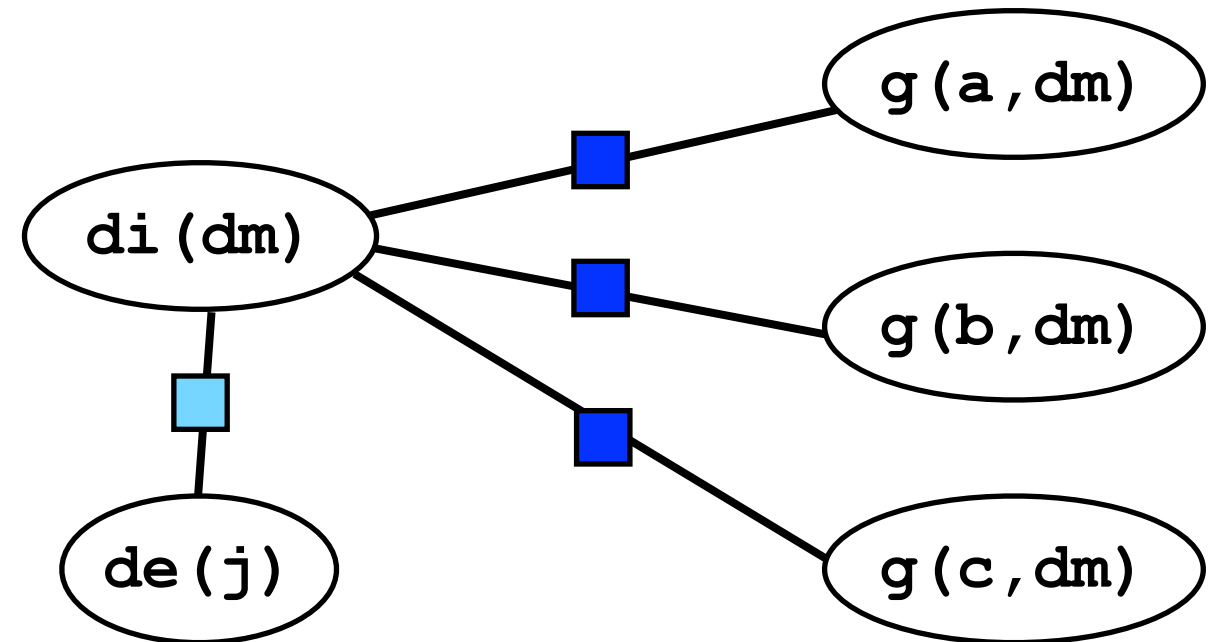
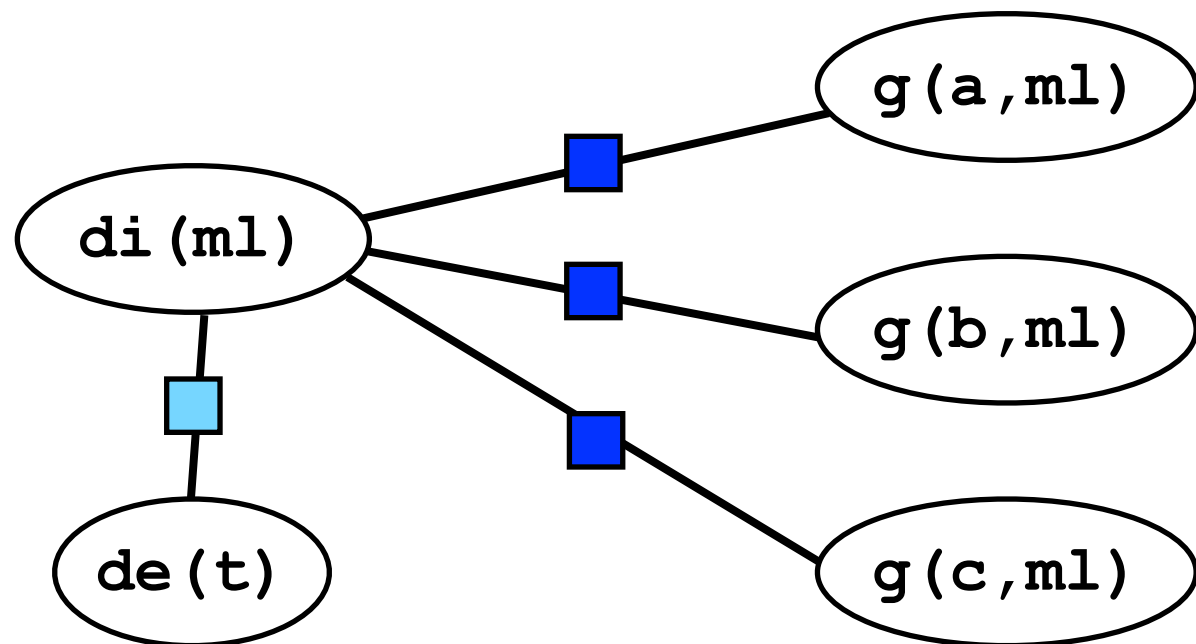
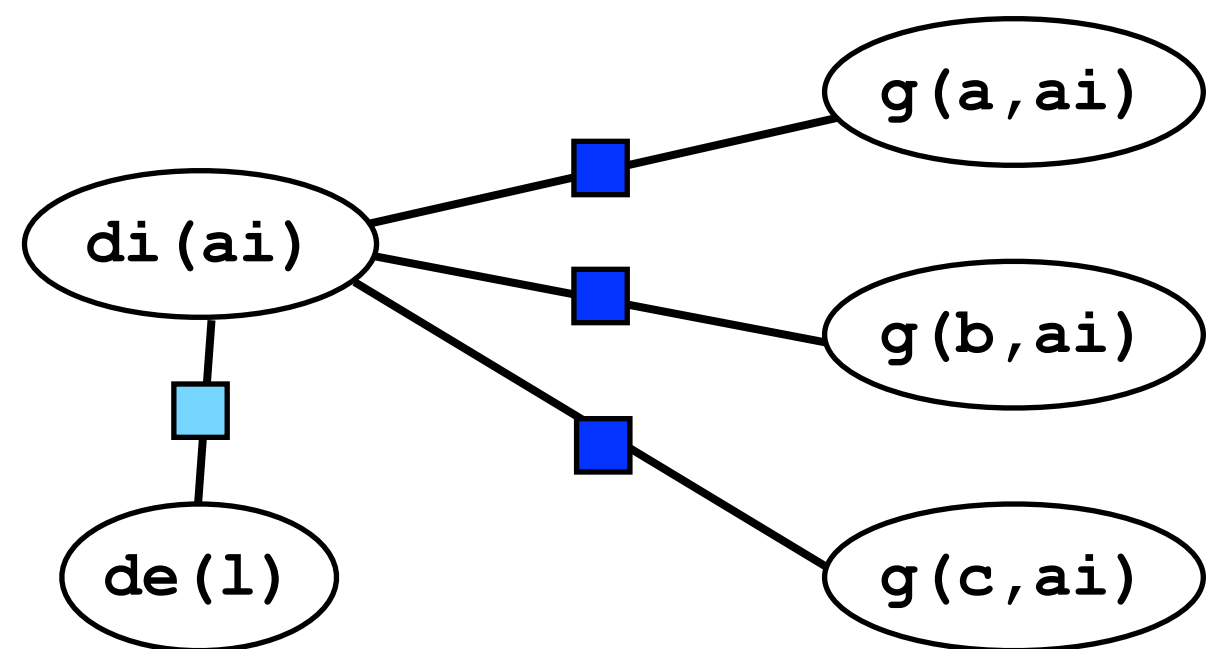
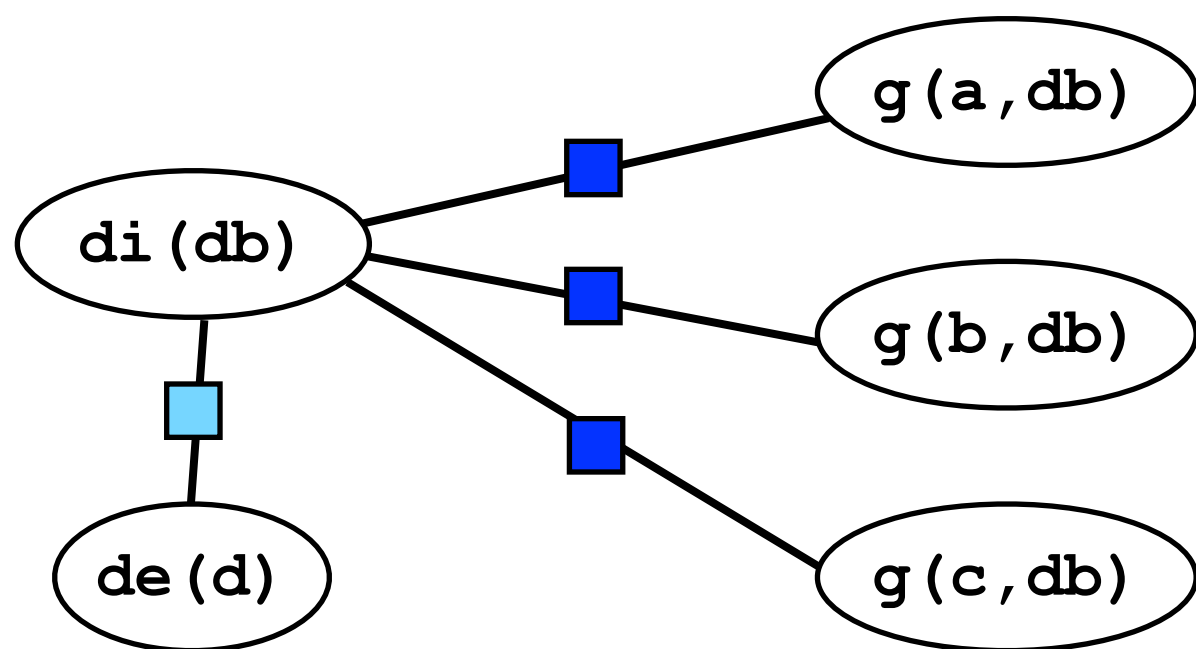




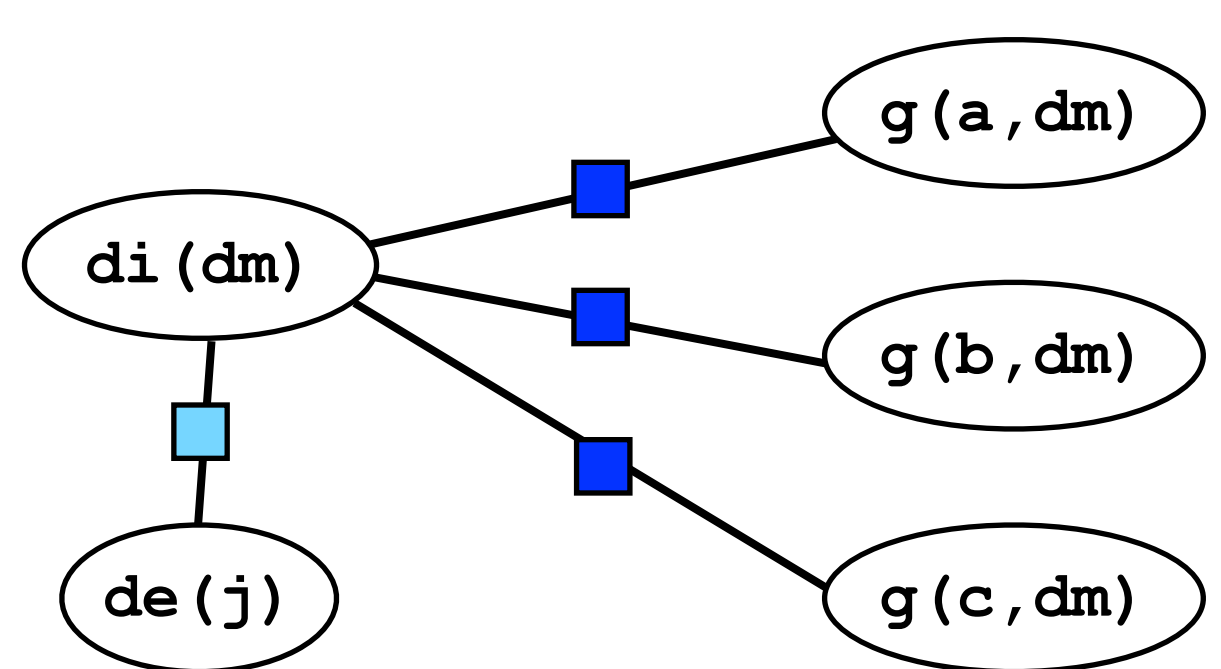
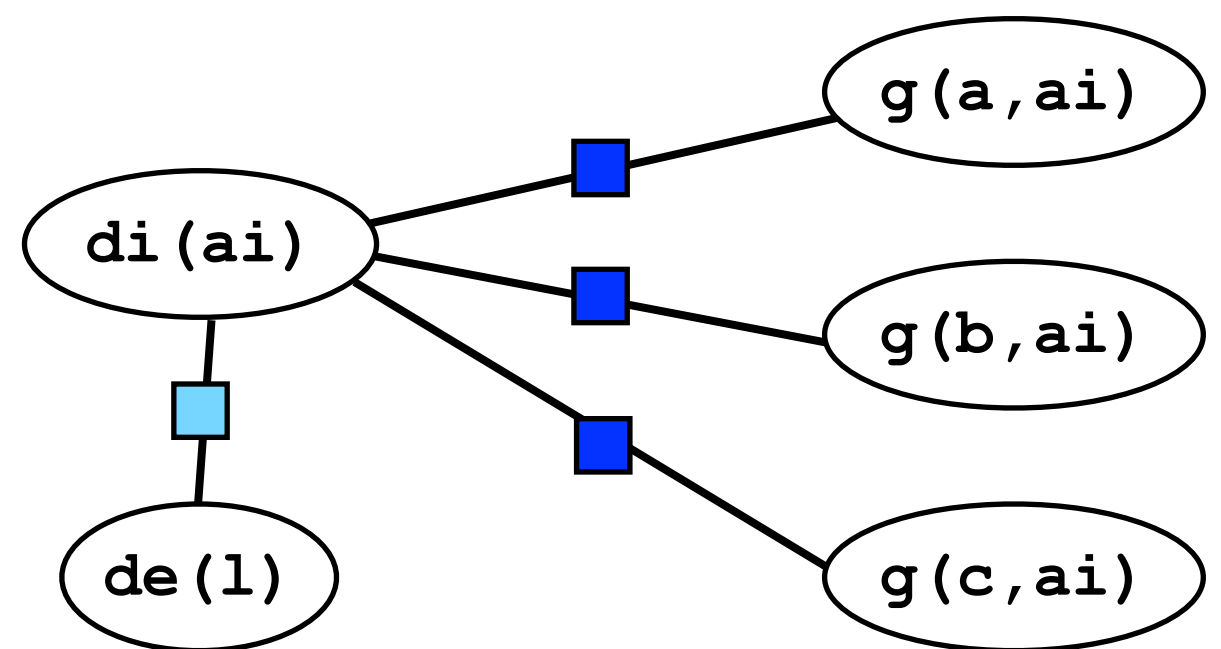
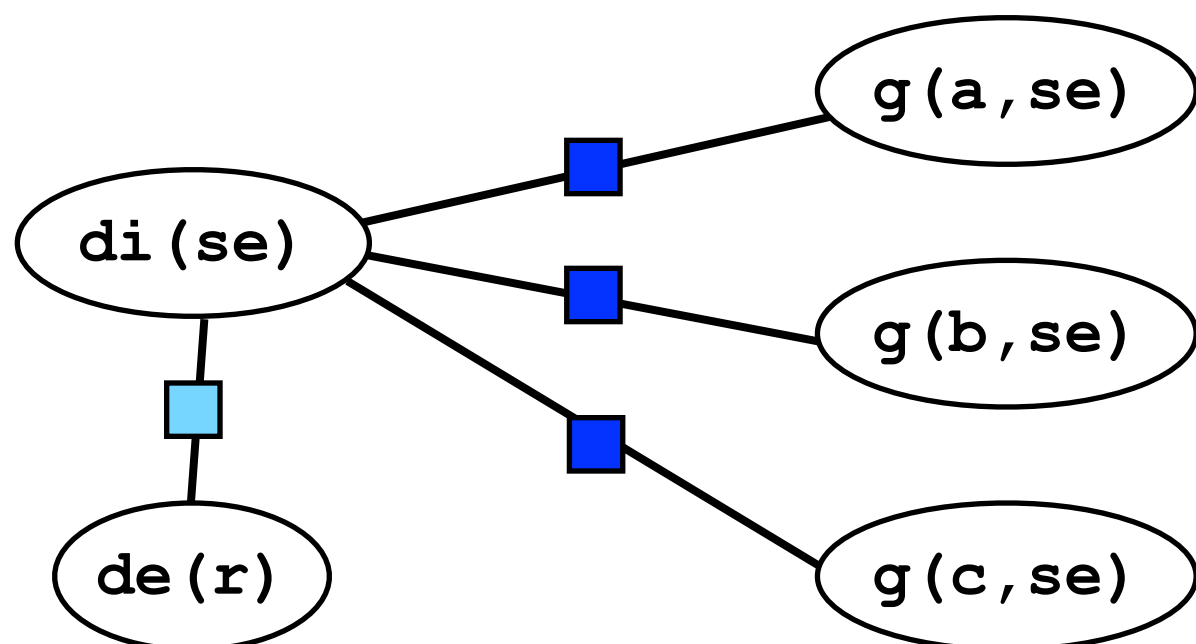
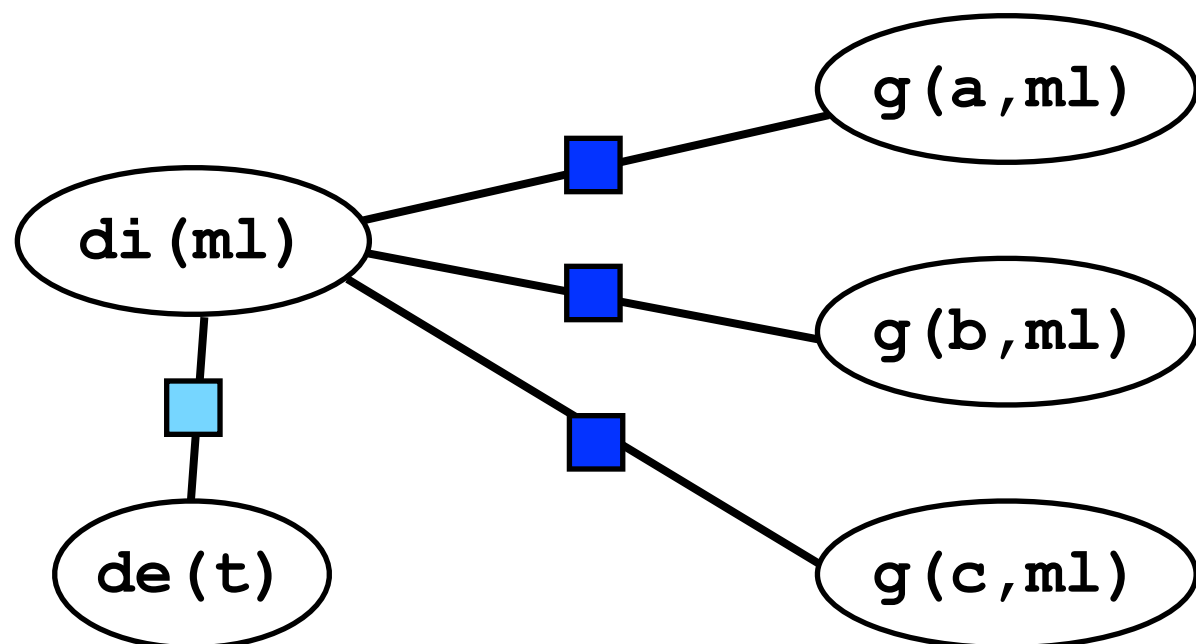
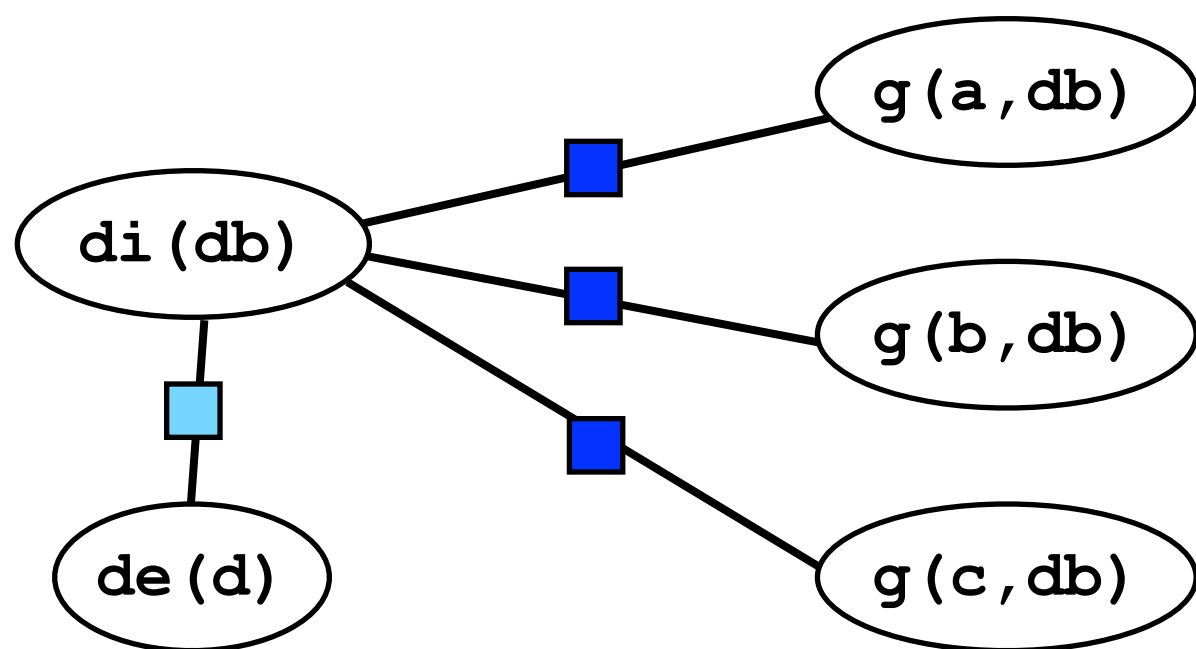
compute probability
distribution over grades for
every student-course pair



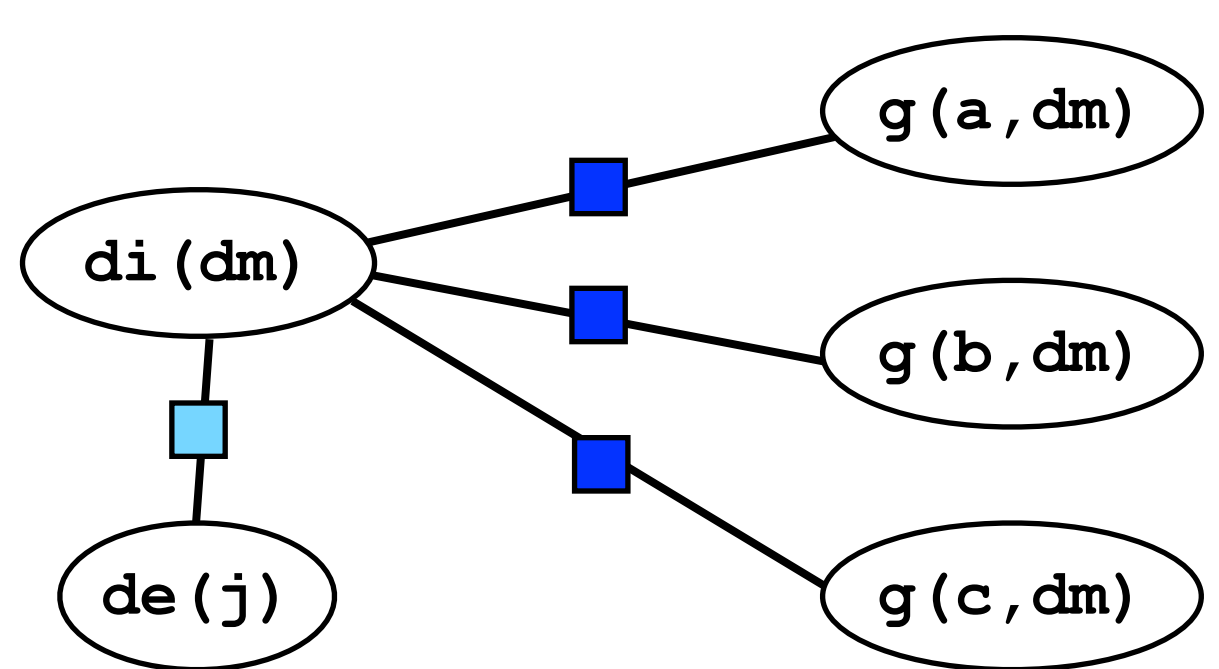
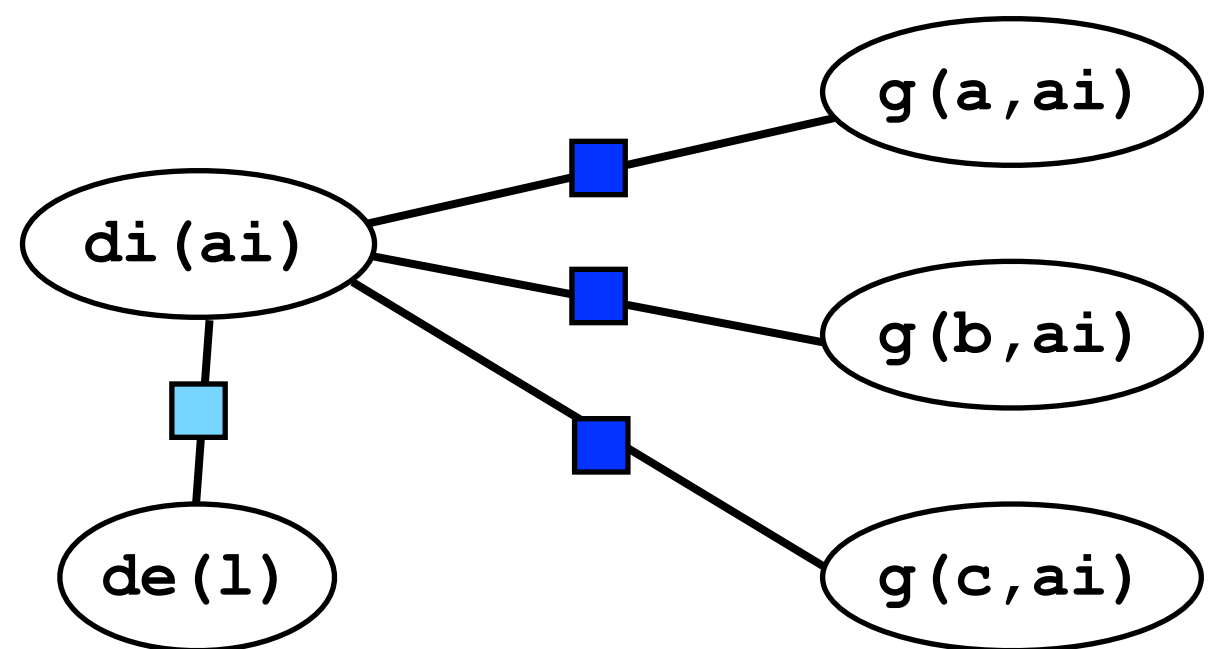
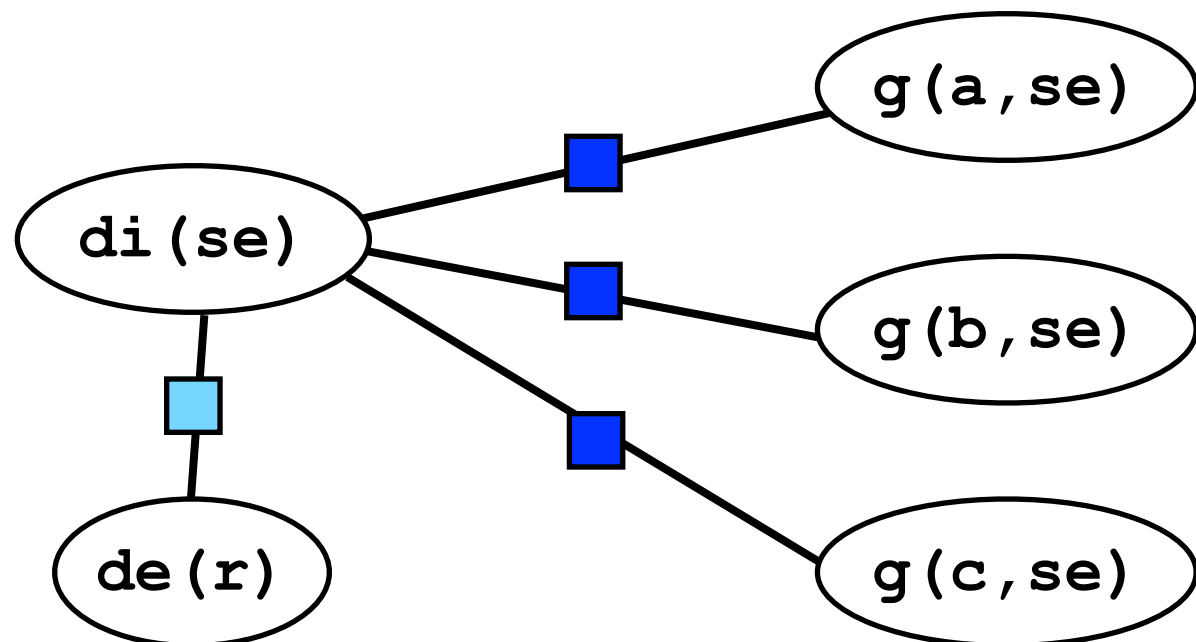
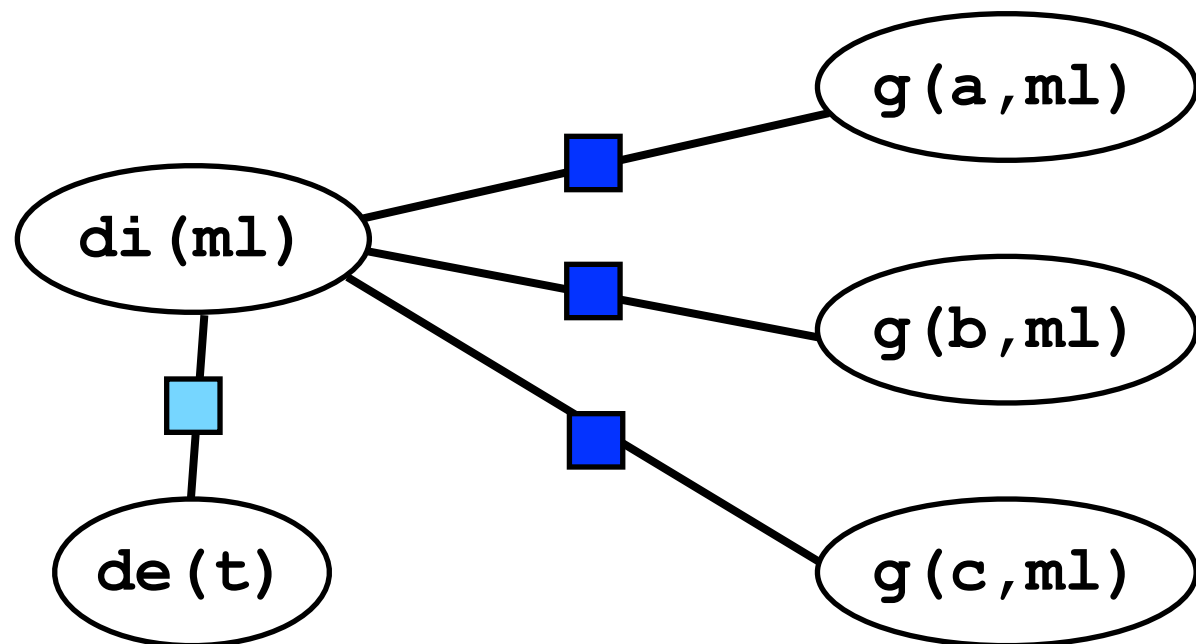
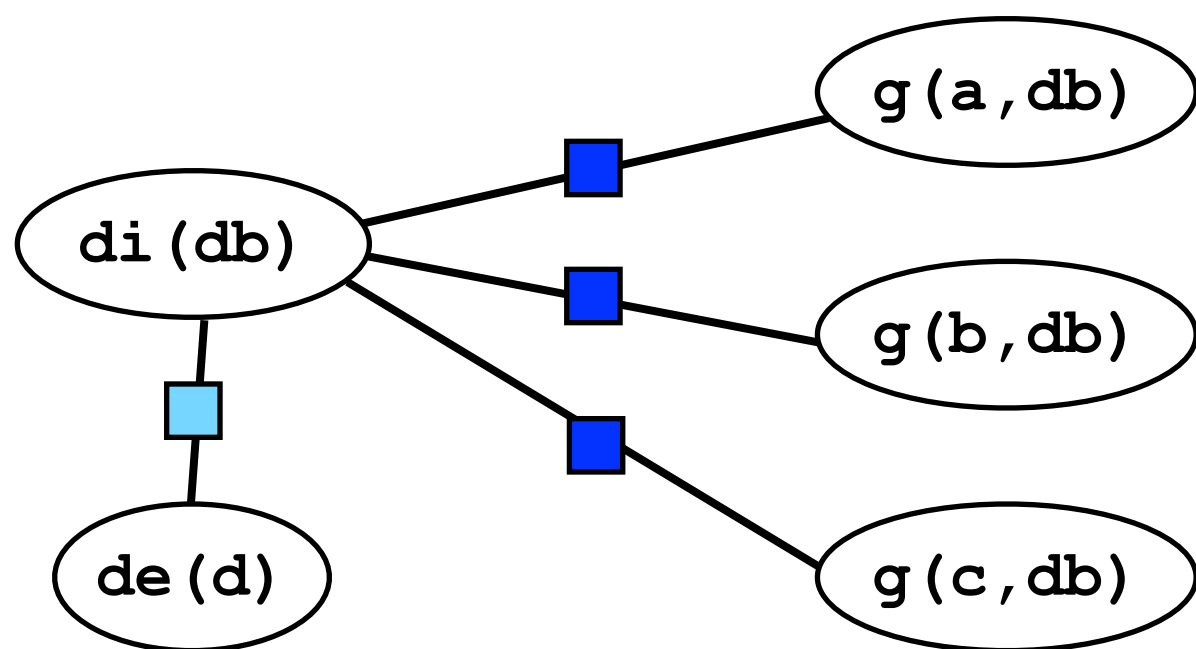
compute probability
distribution over grades for
every student-course pair
same computation for each pair!



what is the probability that
some professor is demanding?



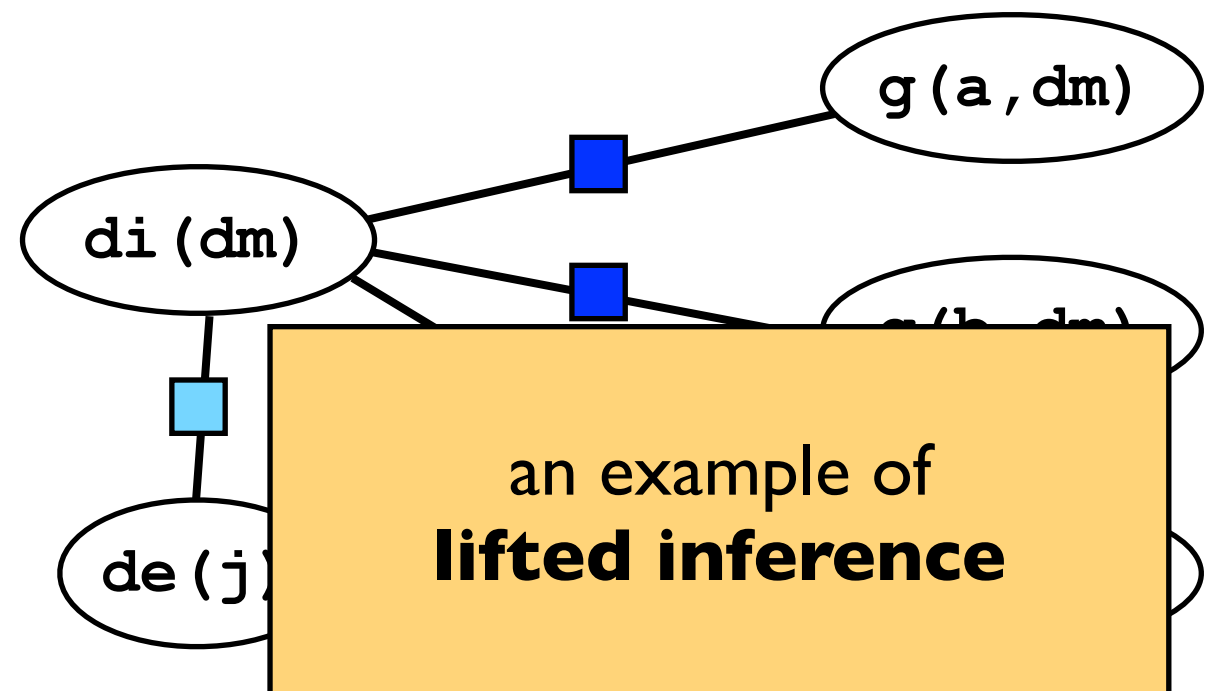
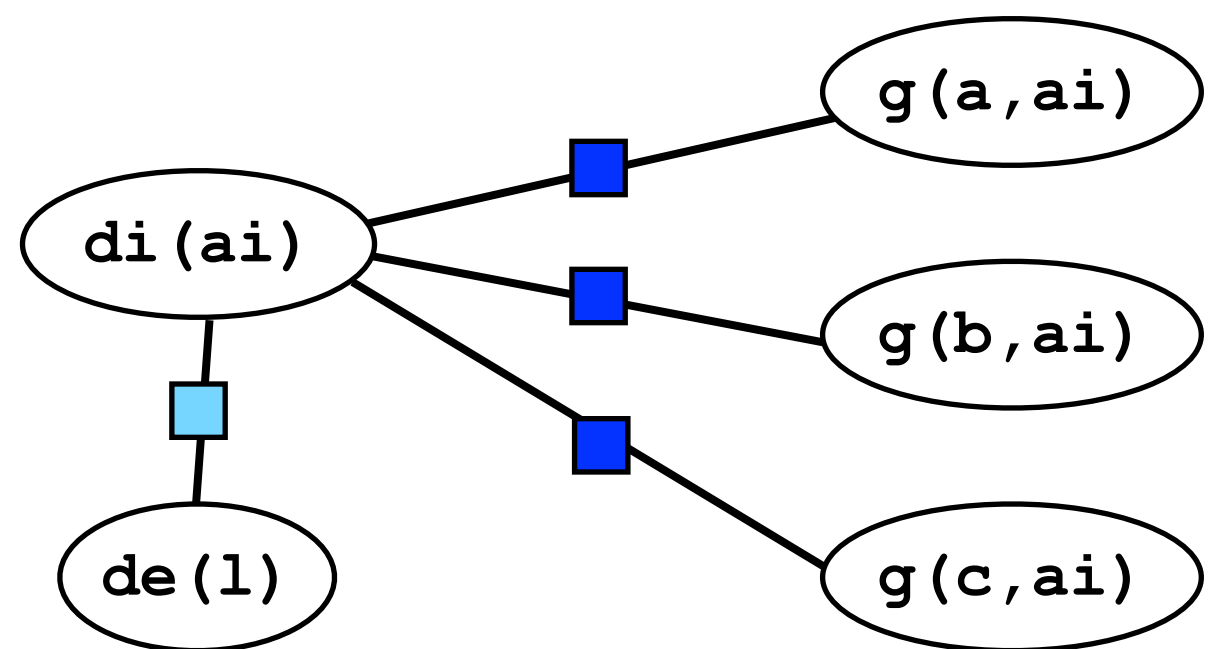
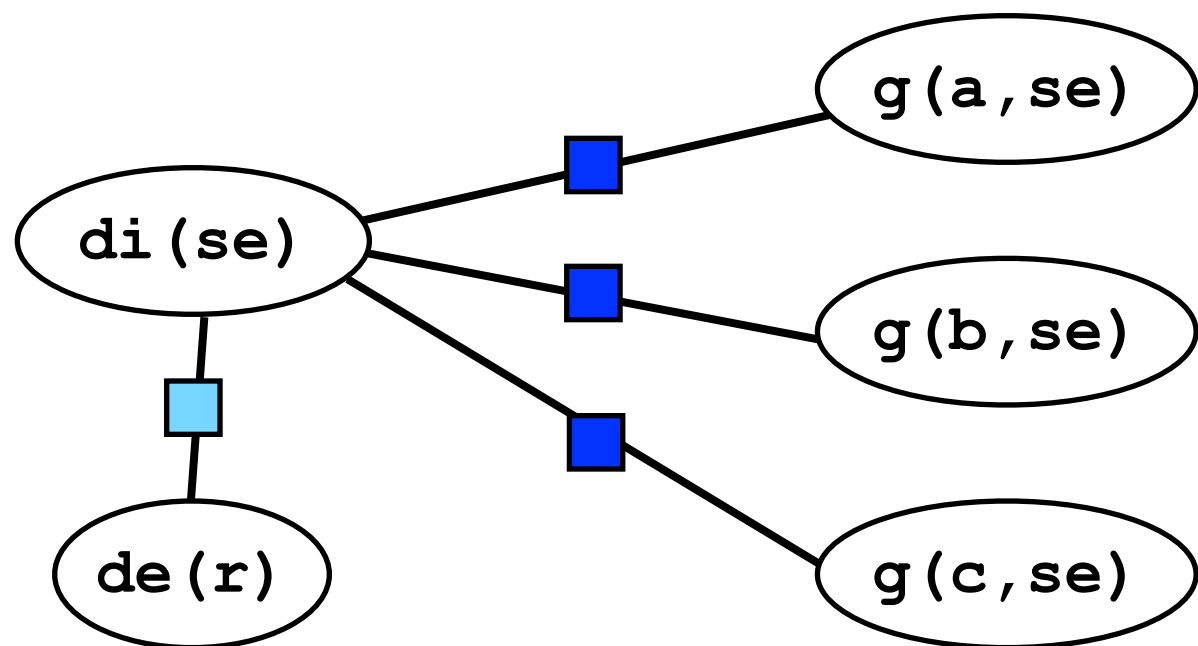
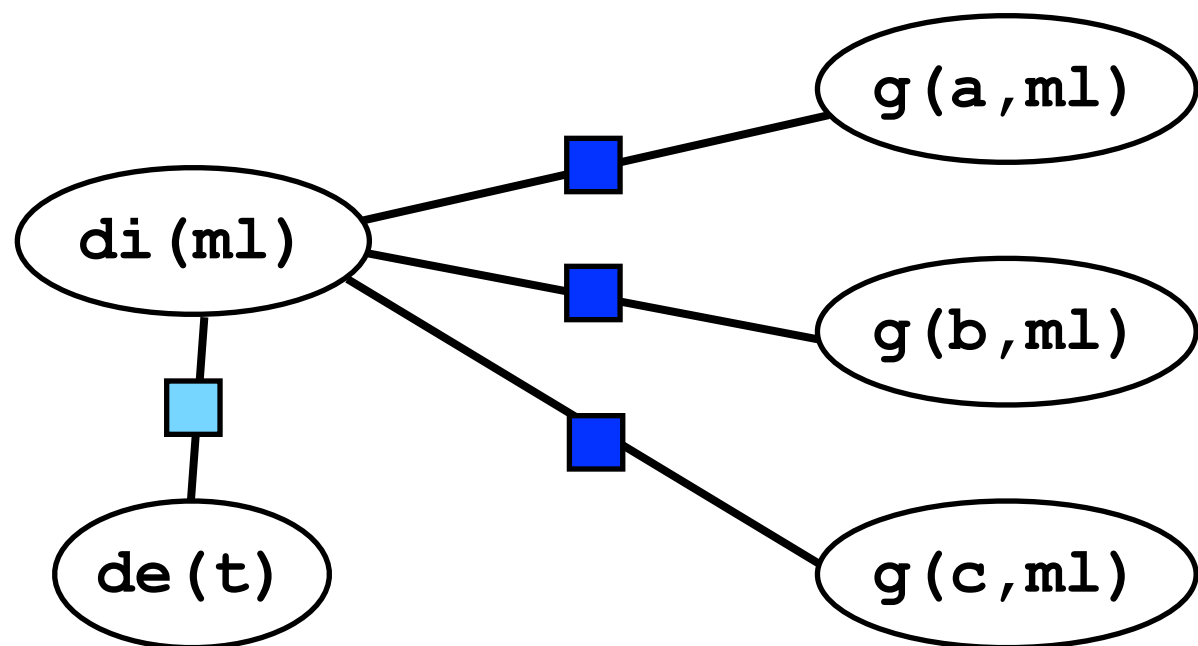
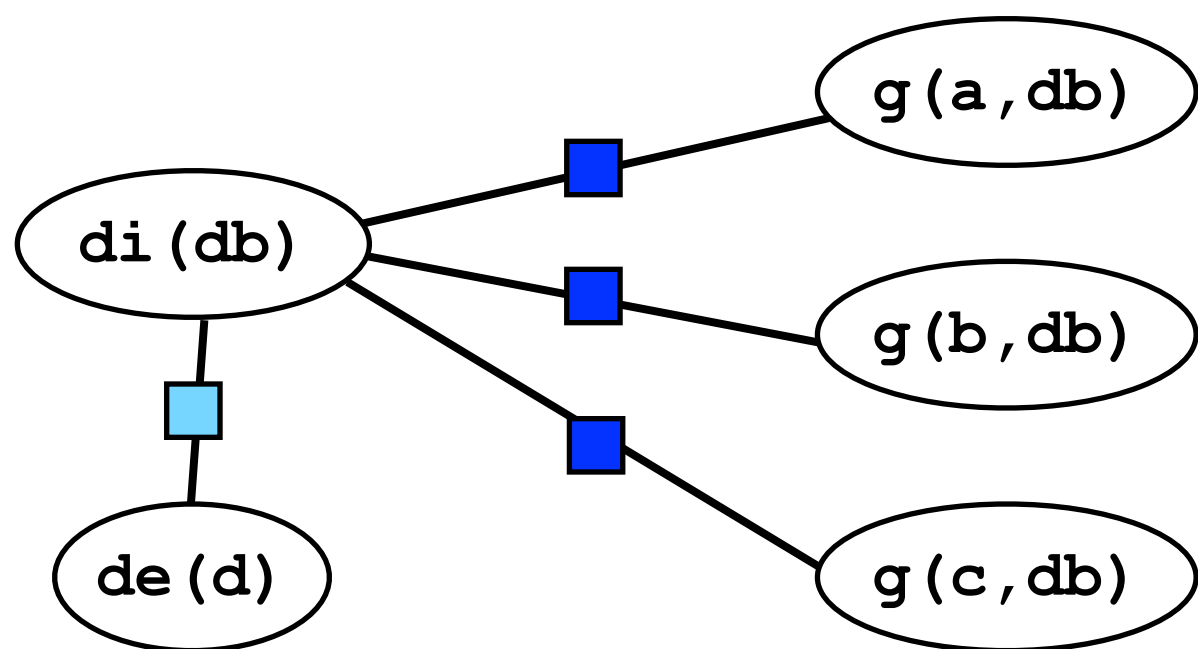
what is the probability that
some professor is demanding?
same computation for each
professor!



what is the probability that
some professor is demanding?

$$1 - \prod_P (1 - P(\text{de}(P))) = 1 - (1 - P(\text{de}(\text{some}P)))^{\#P}$$

$$= 1 - (1 - P(\text{de}(r)))^5$$



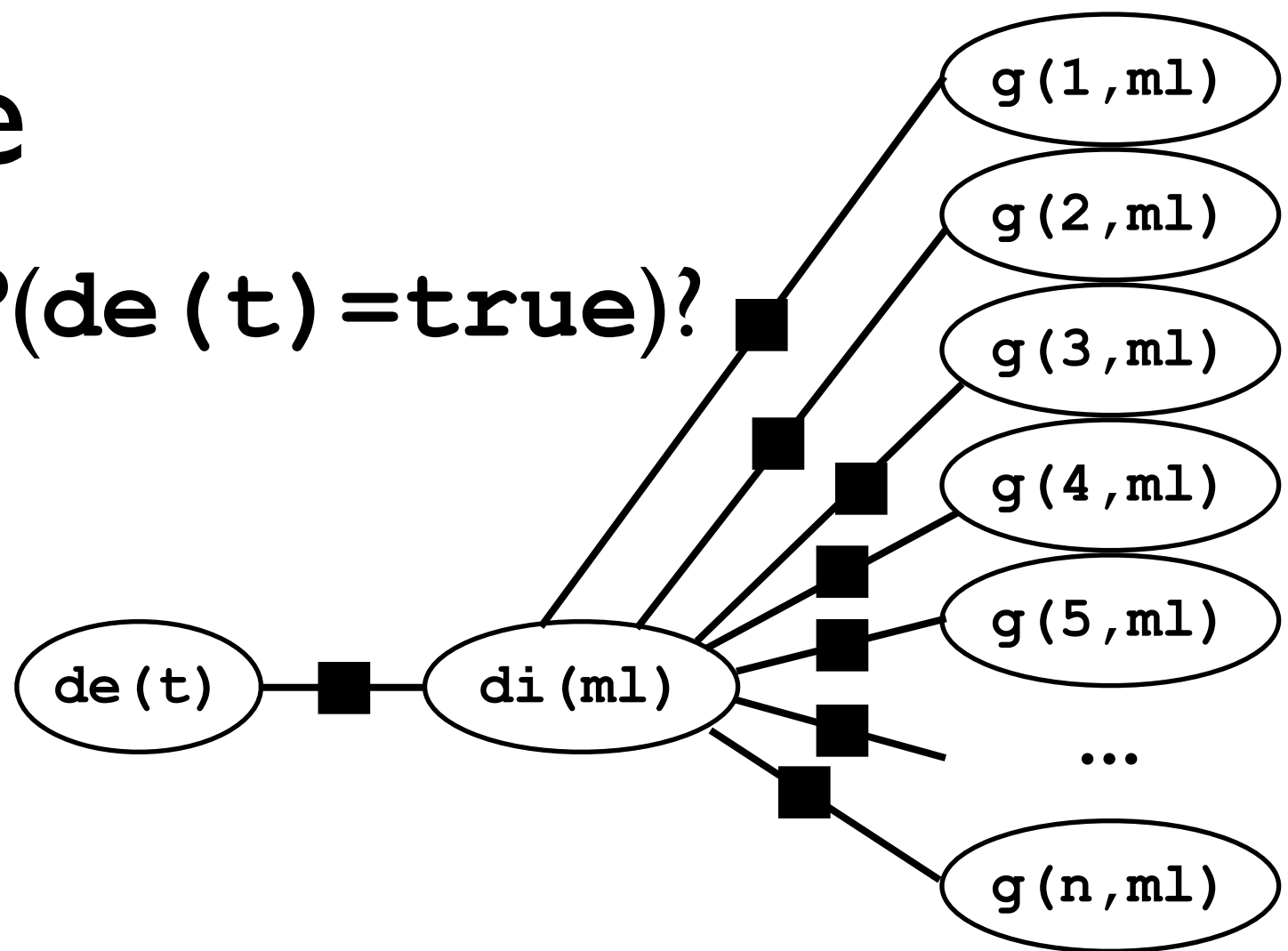
what is the probability that
some professor is demanding?

$$1 - \prod_P (1 - P(\text{de}(P))) = 1 - (1 - P(\text{de}(\text{someP})))^{\#P}$$

$$= 1 - (1 - P(\text{de}(r)))^5$$

Another example

$P(\text{de}(t) = \text{true})?$

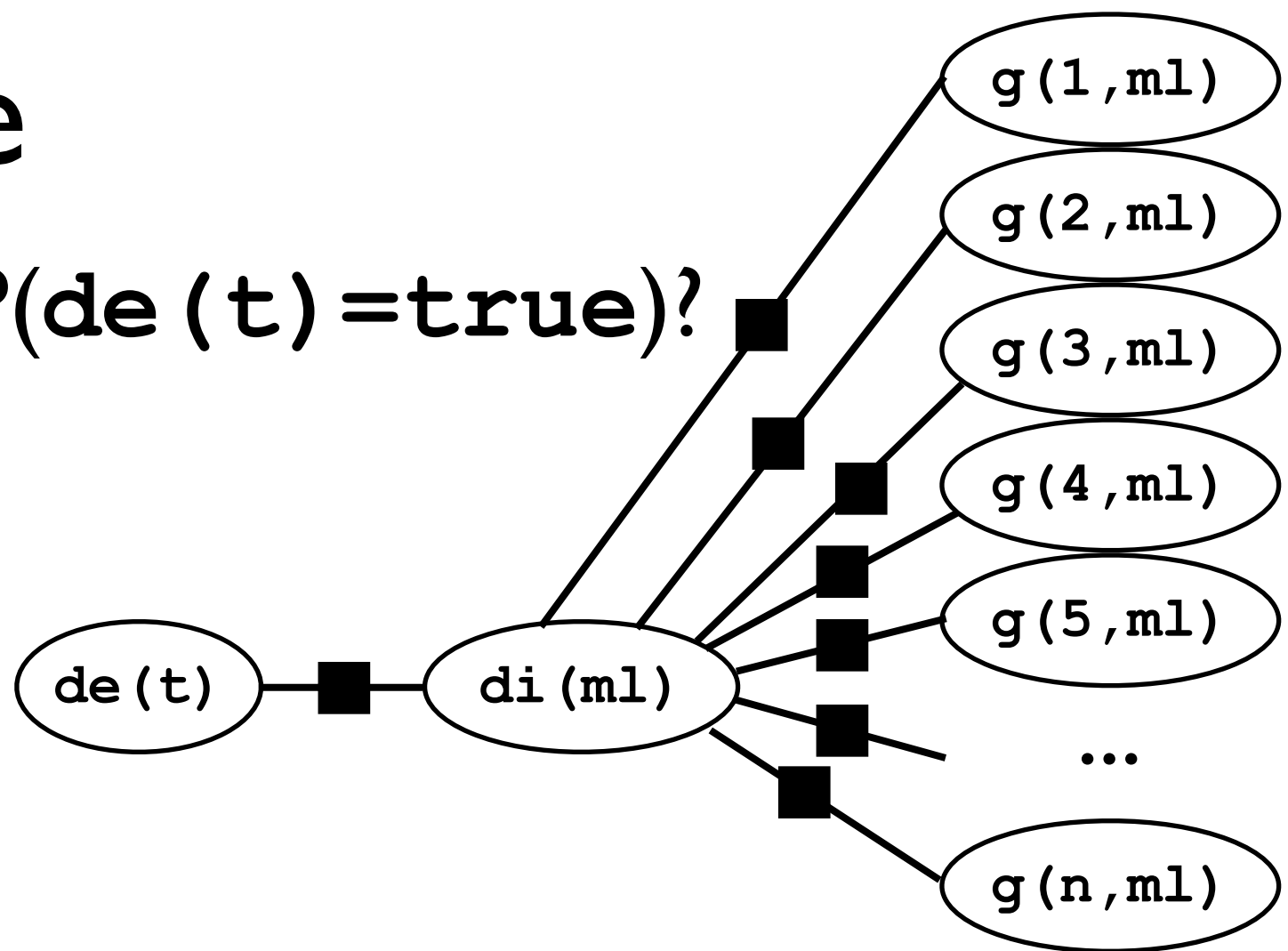


Another example

$$\text{sum } \phi_2(\text{true}, d) \prod_{i=1..n} \phi_1(g_i, d)$$

for all combinations of
difficulty d and students'
grades (g_1, \dots, g_n)

$P(\text{de}(t) = \text{true})?$



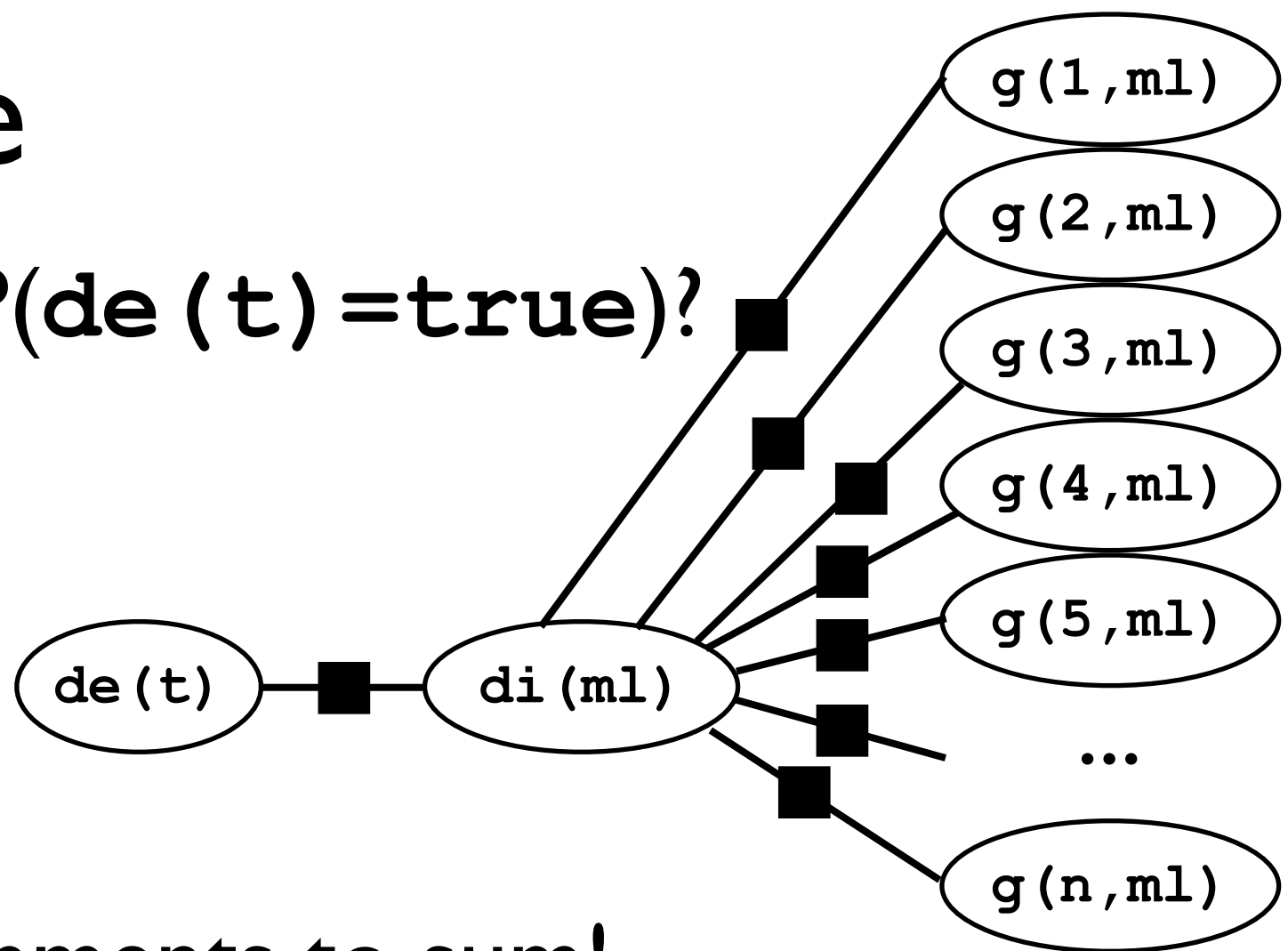
Another example

$$\text{sum } \phi_2(\text{true}, d) \prod_{i=1..n} \phi_1(g_i, d)$$

for all combinations of
difficulty d and students'
grades (g_1, \dots, g_n)

$O(\# \text{grades}^{\# \text{students}})$ assignments to sum!

$P(\text{de}(t) = \text{true})?$



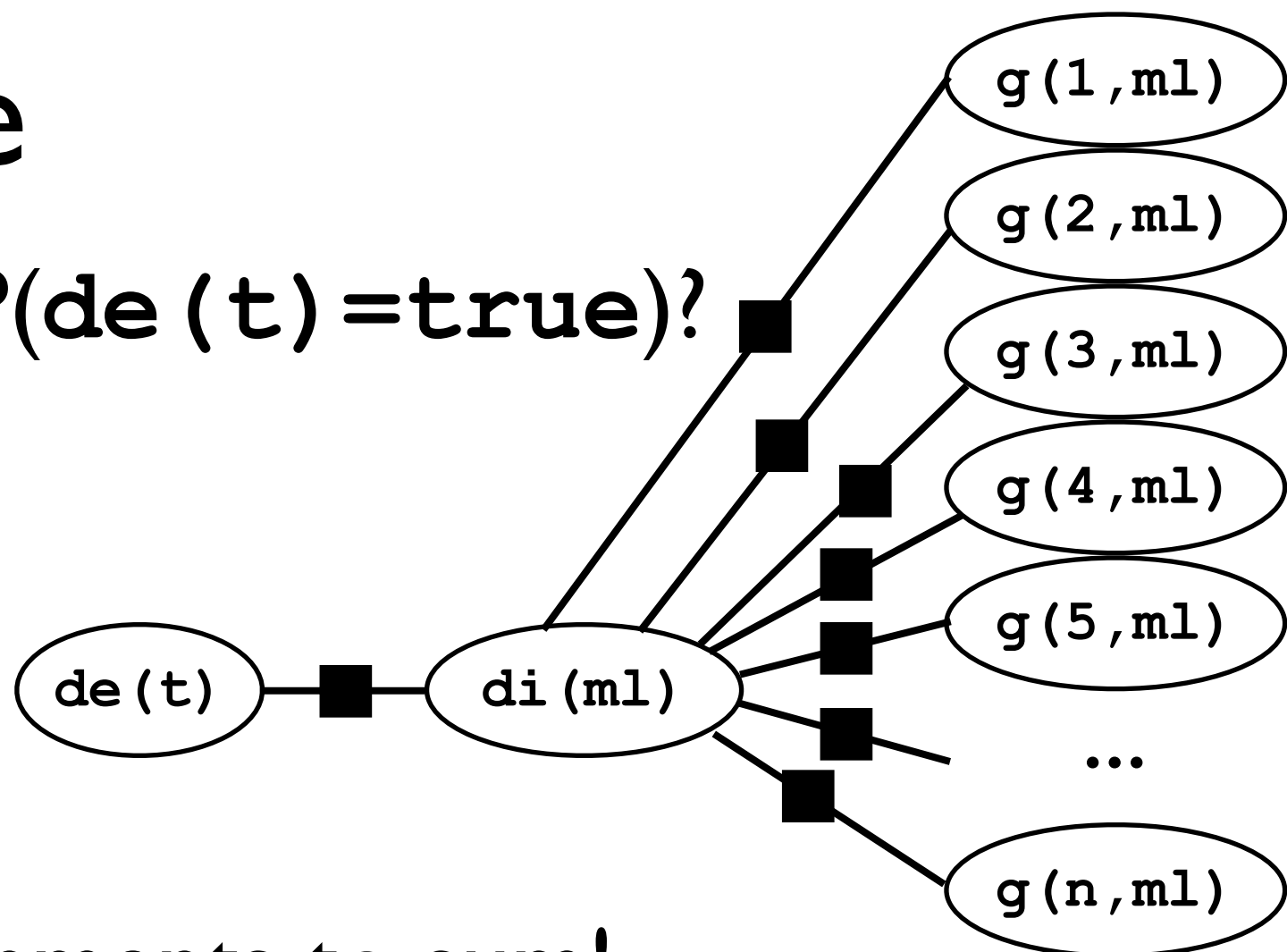
Another example

$$\text{sum } \phi_2(\text{true}, d) \prod_{i=1..n} \phi_1(g_i, d)$$

for all combinations of
difficulty d and students'
grades (g_1, \dots, g_n)

$O(\# \text{grades}^{\# \text{students}})$ assignments to sum!

$P(\text{de}(t) = \text{true})?$



but: identity of students doesn't matter!

sufficient to count how often each grade m_1, \dots, m_k appears

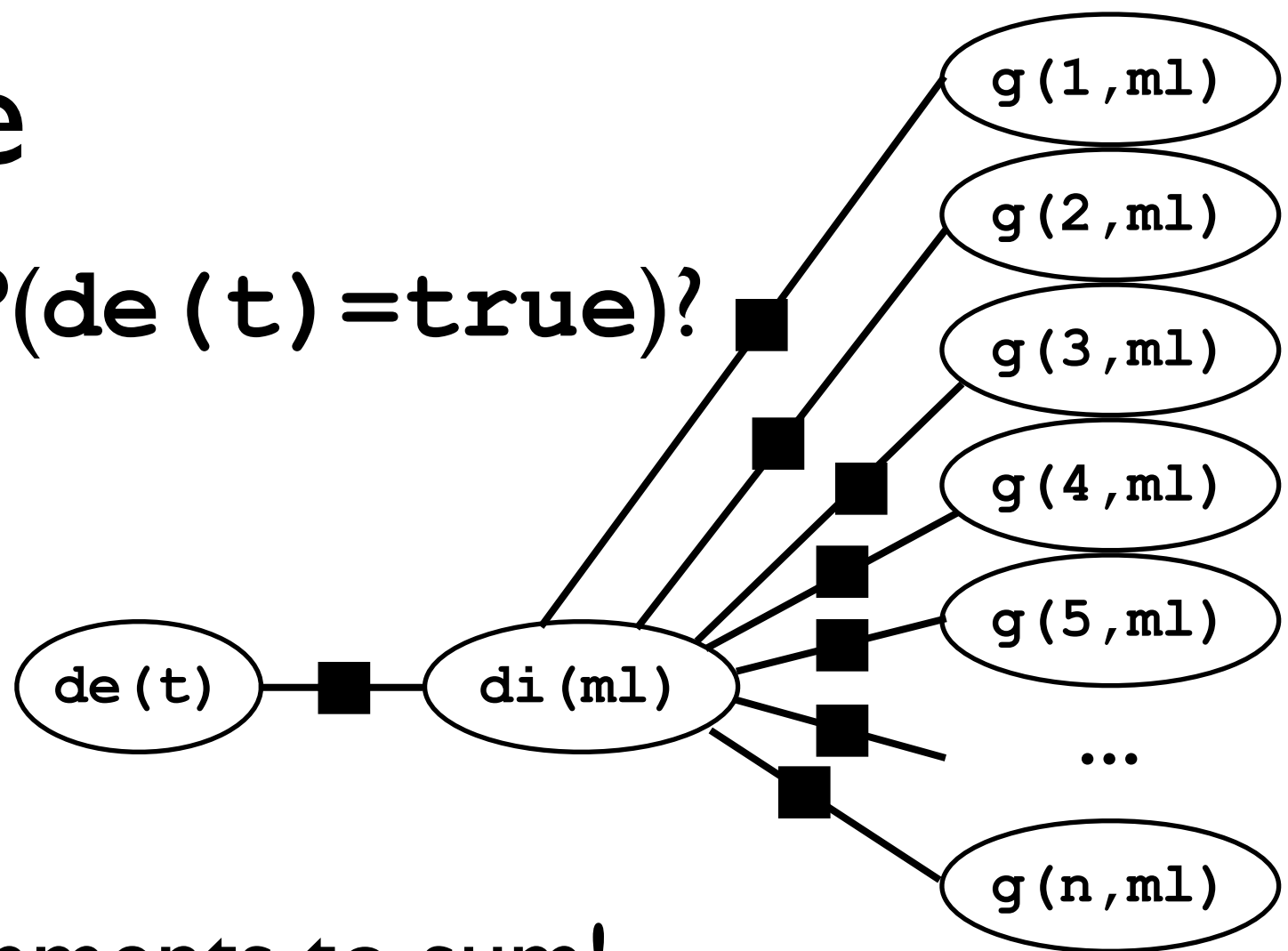
Another example

$$\text{sum } \phi_2(\text{true}, d) \prod_{i=1..n} \phi_1(g_i, d)$$

for all combinations of
difficulty d and students'
grades (g_1, \dots, g_n)

$O(\# \text{grades}^{\# \text{students}})$ assignments to sum!

$P(\text{de}(t) = \text{true})?$



but: identity of students doesn't matter!

sufficient to count how often each grade m_1, \dots, m_k appears

$$\text{sum } \phi_2(\text{true}, d) \prod_{i=1..k} \phi_1(m_i, d)^{\#m_i} \text{ instead}$$

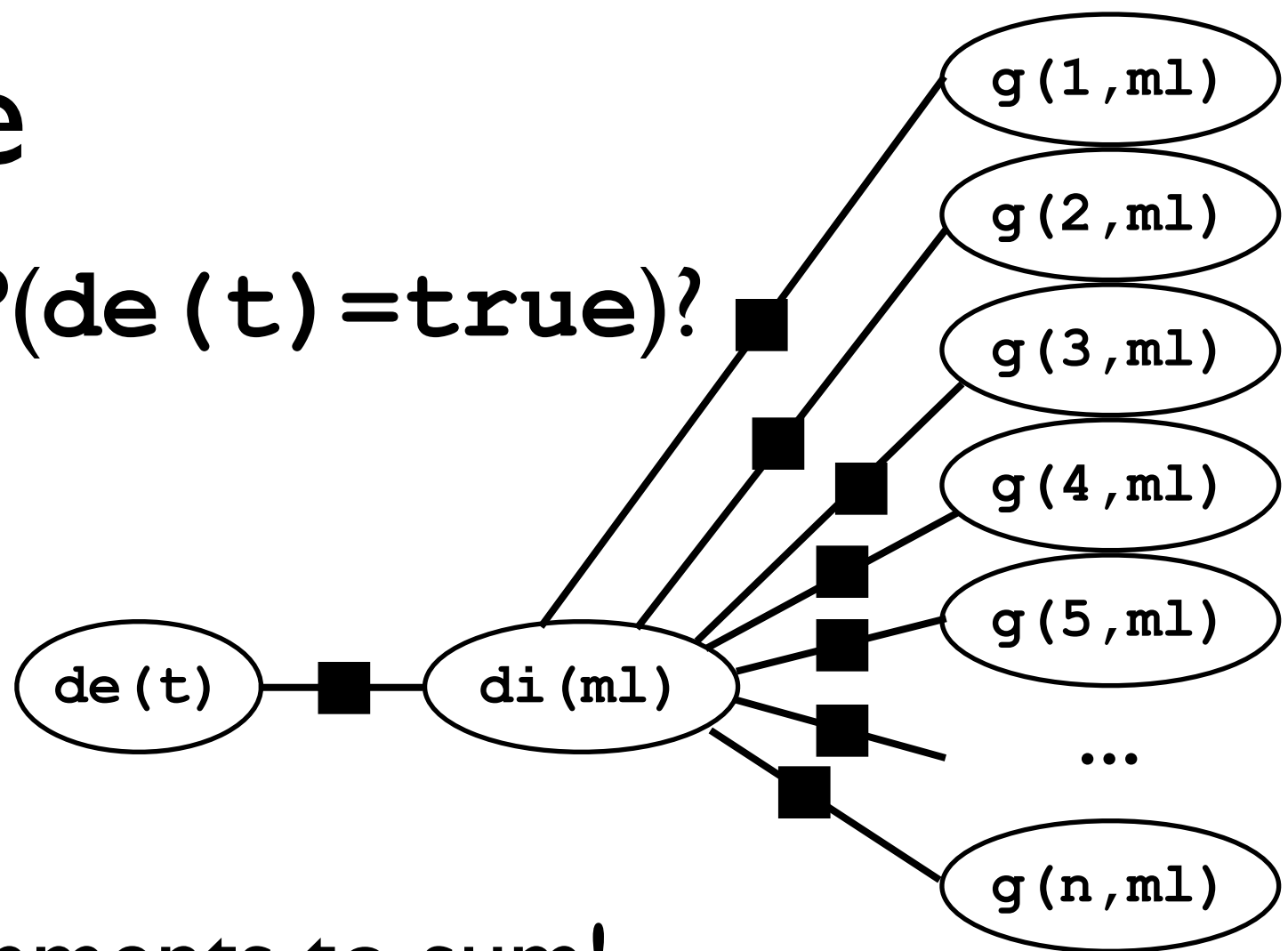
Another example

$$\text{sum } \phi_2(\text{true}, d) \prod_{i=1..n} \phi_1(g_i, d)$$

for all combinations of
difficulty d and students'
grades (g_1, \dots, g_n)

$O(\# \text{grades}^{\# \text{students}})$ assignments to sum!

$P(\text{de}(t) = \text{true})?$



but: identity of students doesn't matter!

sufficient to count how often each grade m_1, \dots, m_k appears

$$\text{sum } \phi_2(\text{true}, d) \prod_{i=1..k} \phi_1(m_i, d)^{\#m_i} \text{ instead}$$

e.g., $k=3, n=15$: $> 14\text{M}$ grade vectors vs 136 count vectors

Lifted Inference

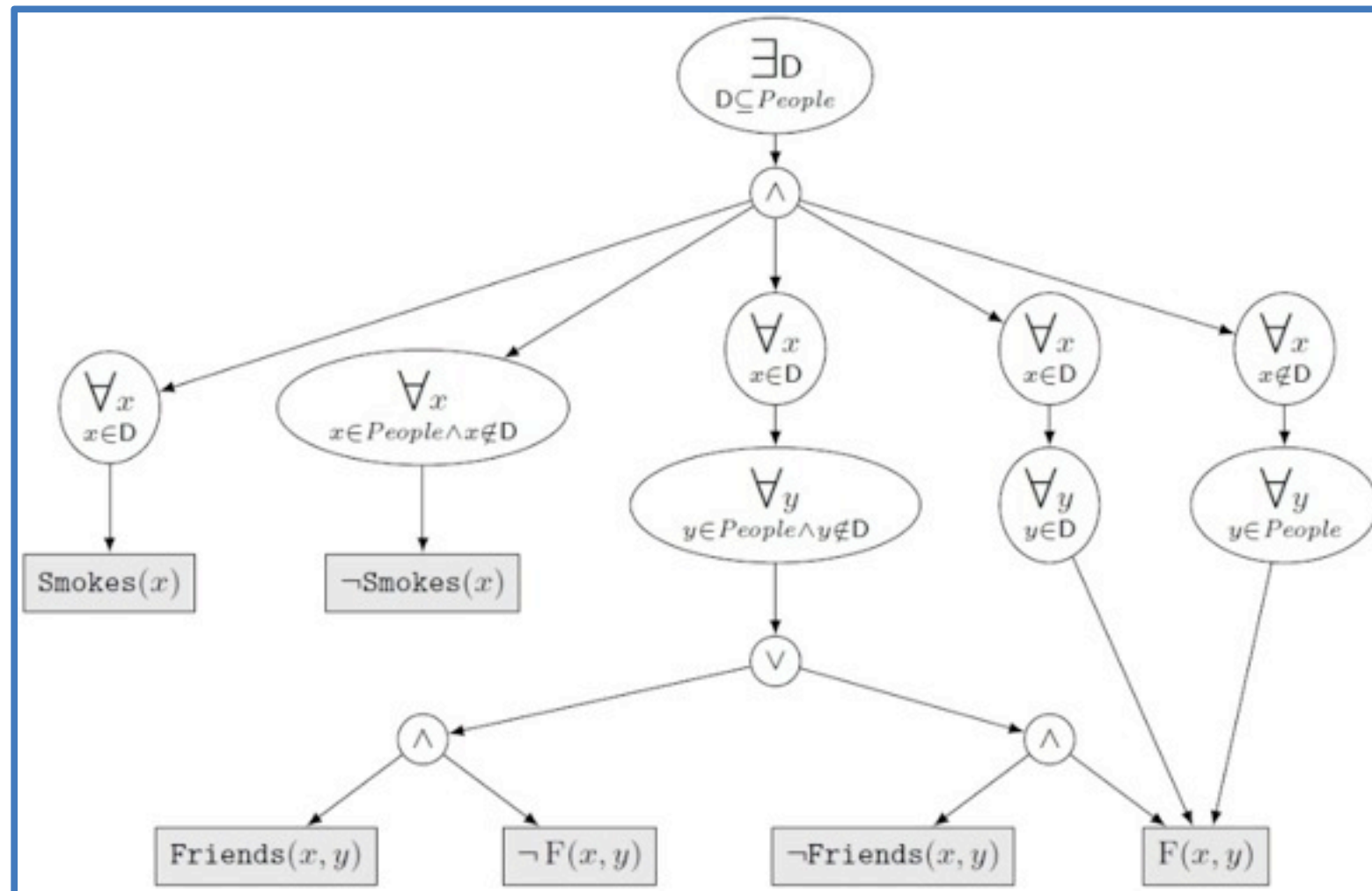
- exploiting symmetries & repeated structure
- reasoning on first order level as much as possible
- aiming at independence from number of objects
- approximation: grouping similar computations
- very active research area

Domain-lifted Inference

[Van den Broeck, NIPS 11]

- a **formal** definition of lifted inference:
probabilistic inference in time polynomial in
the **number of objects** in the domain
- completeness result: probabilistic models
with up to two logical variables per clause
are domain-liftable

Example: Weighted First-Order Model Counting (WFOMC)



First-Order d-DNNF Circuit

$\text{Smokes} \rightarrow 1$
 $\neg \text{Smokes} \rightarrow 1$
 $\text{Friends} \rightarrow 1$
 $\neg \text{Friends} \rightarrow 1$
 $F \rightarrow \exp(3.14)$
 $\neg F \rightarrow 1$

Weight Function

Alice
 Bob
 Charlie

Domain

Weighted First-Order Model Count is **1479.85**

Lifted Inference for PLP

- LP²: translate ProbLog program to Prolog Factor Language & run lifted variable elimination (GC-FOVE with new operator) [Bellodi et al, ICLP 14]
- new skolemization procedure that is sound for WFOMC [Van den Broeck et al, KR 14]
- thus: ProbLog programs with up to two logical variables per clause (and tied parameters) are domain-liftable

PROBABILISTIC LOGIC PROGRAM

0.1 :: Attends(x).
0.3 :: Series :- Attends(x).



Series $\Leftrightarrow \exists x, \text{Attends}(x) \wedge \text{ToSeries}(x).$

$w(\text{Attends}) = 0.1$ $w(\text{ToSeries}) = 0.3$

$\bar{w}(\text{Attends}) = 0.9$ $\bar{w}(\text{ToSeries}) = 0.7$

$w(\cdot) = 1$ $\bar{w}(\cdot) = 1$

PROBABILISTIC LOGIC PROGRAM

$$\text{Series} \Leftrightarrow \exists x, \text{Attends}(x) \wedge \text{ToSeries}(x).$$



$$\text{Series} \Leftrightarrow Z$$

$$\forall x, \quad Z \vee \neg[\text{Attends}(x) \wedge \text{ToSeries}(x)]$$

$$Z \vee S$$

$$\forall x, \quad S \vee \neg[\text{Attends}(x) \wedge \text{ToSeries}(x)]$$

$$w(\text{Attends}) = 0.1 \quad w(\text{ToSeries}) = 0.3$$

$$\bar{w}(\text{Attends}) = 0.9 \quad \bar{w}(\text{ToSeries}) = 0.7$$

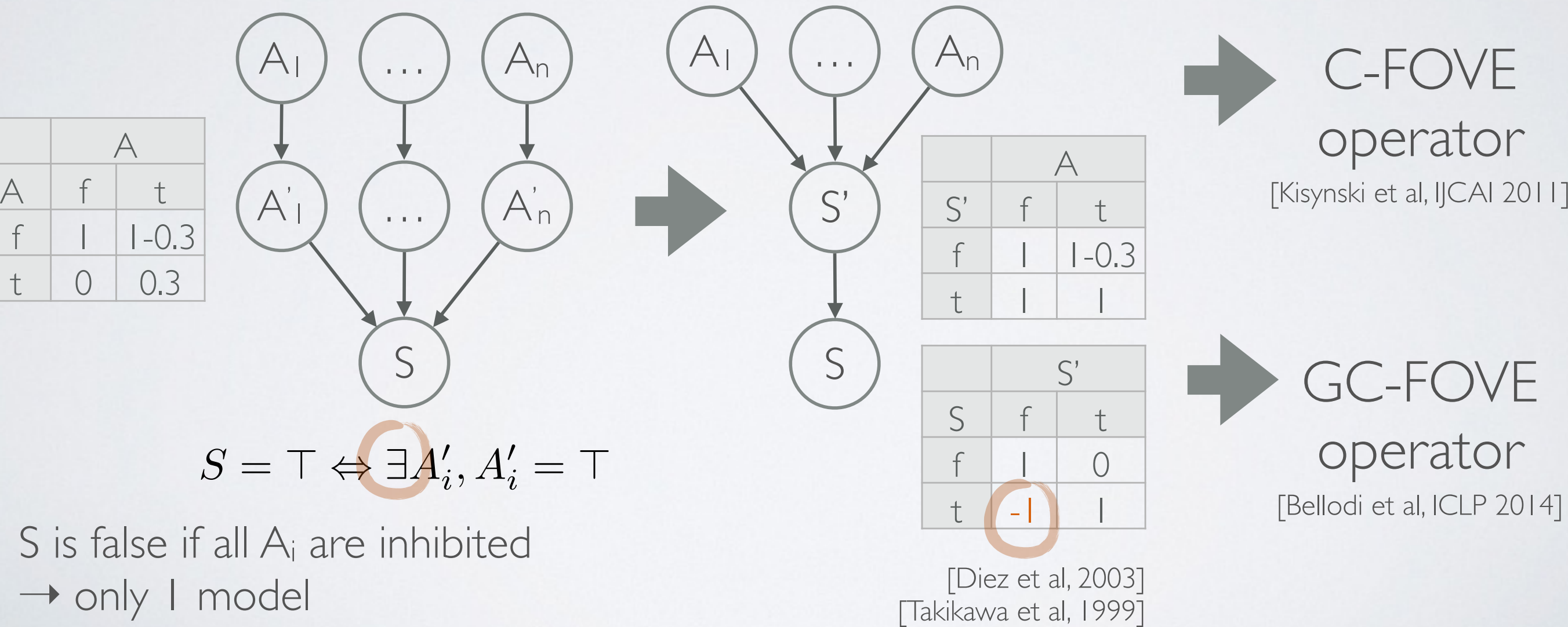
$$\bar{w}(S) = -1$$

$$w(\cdot) = 1 \quad \bar{w}(\cdot) = 1$$

NOISY-OR

Multiplicative factorisation (or tensor rank-one decomposition) for noisy-OR is a special case of the Skolemization procedure

0.1::Attends(x).
0.3::Series :- Attends(x).



$S = \top \Leftrightarrow \exists A'_i, A'_i = \top$

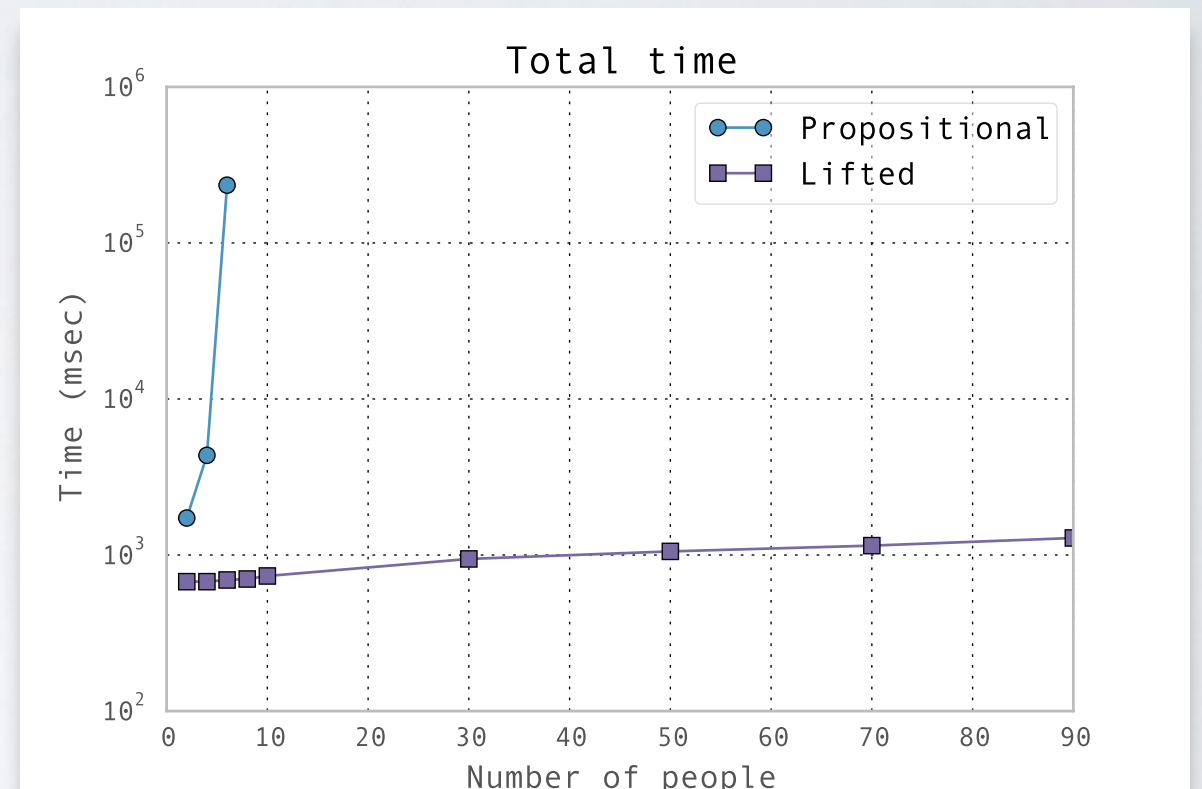
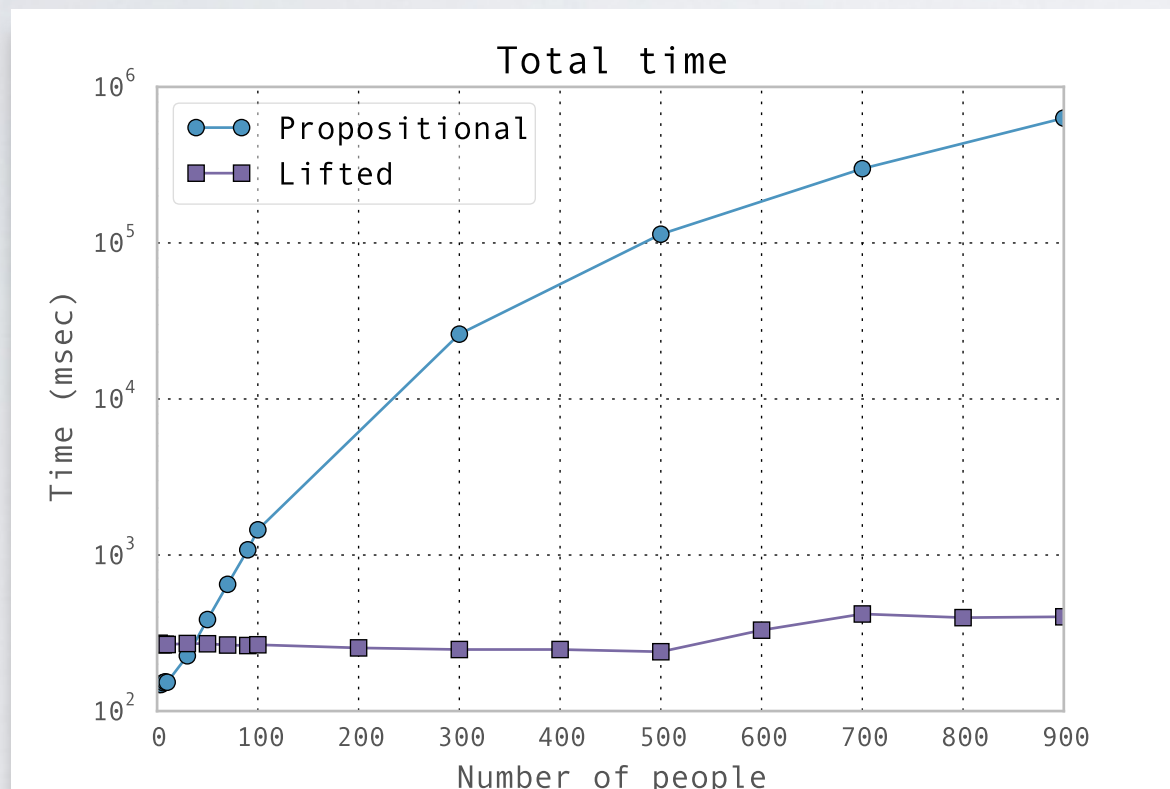
S is false if all A_i are inhibited
→ only 1 model
Relax for this model and then compensate

[Diez et al, 2003]
[Takikawa et al, 1999]

EXPERIMENTS

0.1 :: Attends(x).

0.3 :: Series :- Attends(x).



0.1 :: Attends(x).

0.3 :: Series :- Attends(x), Coauthor(x, y), Attends(y).

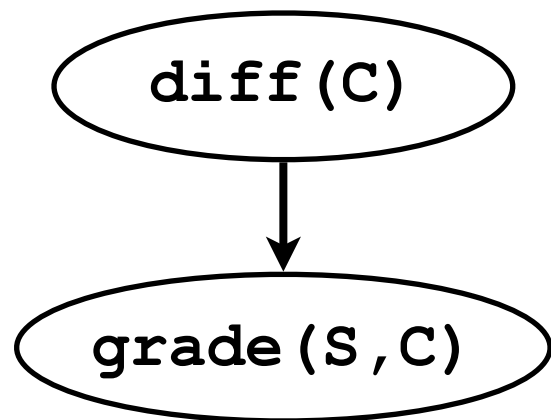
Lifted Graphical Models:

Parameter Learning

- **data** = interpretations on ground level
- **estimate** parameters on first-order level
taking all instances of same par-RV together

Lifted Graphical Models: Parameter Learning

- **data** = interpretations on ground level
- **estimate** parameters on first-order level
taking all instances of same par-RV together



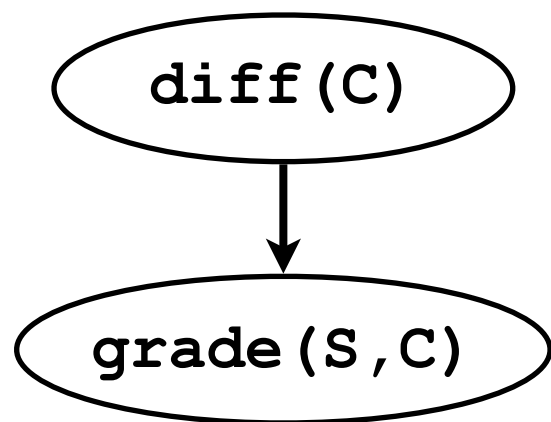
`diff(m1)=t`
`diff(db)=f`
`grade(a,m1)=h`
`grade(b,m1)=1`
`grade(a,db)=1`
`grade(c,db)=h`

`diff(se)=f`
`diff(ds)=t`
`grade(b,se)=1`
`grade(c,se)=1`
`grade(d,ds)=h`
`grade(g,ds)=1`

`diff(ai)=t`
`grade(f,ai)=1`
`grade(e,ai)=h`
`grade(g,ai)=h`
`grade(d,ai)=h`
`grade(a,ai)=1`

Lifted Graphical Models: Parameter Learning

- **data** = interpretations on ground level
- **estimate** parameters on first-order level taking all instances of same par-RV together



`diff(m1)=t`
`diff(db)=f`
`grade(a,m1)=h`
`grade(b,m1)=1`
`grade(a,db)=1`
`grade(c,db)=h`

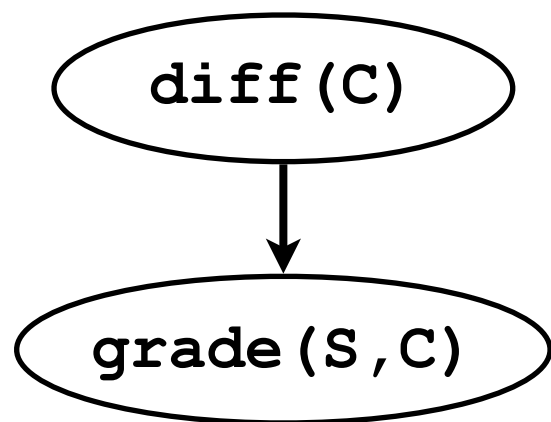
`diff(se)=f`
`diff(ds)=t`
`grade(b,se)=1`
`grade(c,se)=1`
`grade(d,ds)=h`
`grade(g,ds)=1`

`diff(ai)=t`
`grade(f,ai)=1`
`grade(e,ai)=h`
`grade(g,ai)=h`
`grade(d,ai)=h`
`grade(a,ai)=1`

$$P(\text{grade}=h | \text{diff}=t) =$$

Lifted Graphical Models: Parameter Learning

- **data** = interpretations on ground level
- **estimate** parameters on first-order level taking all instances of same par-RV together



`diff(m1)=t`
`diff(db)=f`
`grade(a,m1)=h`
`grade(b,m1)=1`
`grade(a,db)=1`
`grade(c,db)=h`

`diff(se)=f`
`diff(ds)=t`
`grade(b,se)=1`
`grade(c,se)=1`
`grade(d,ds)=h`
`grade(g,ds)=1`

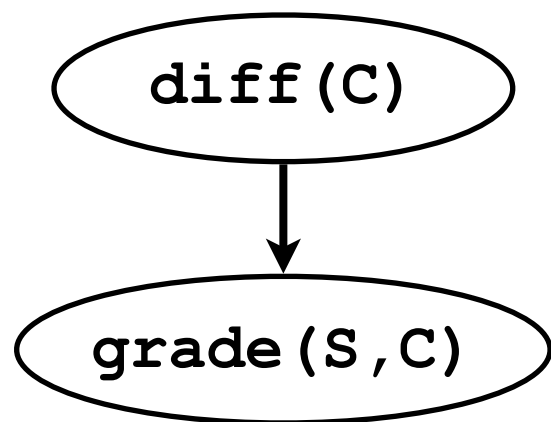
`diff(ai)=t`
`grade(f,ai)=1`
`grade(e,ai)=h`
`grade(g,ai)=h`
`grade(d,ai)=h`
`grade(a,ai)=1`

$$P(\text{grade}=h | \text{diff}=t) = 5/9$$

Lifted Graphical Models:

Parameter Learning

- **data** = interpretations on ground level
- **estimate** parameters on first-order level taking all instances of same par-RV together



`diff(m1)=t`
`diff(db)=f`
`grade(a,m1)=h`
`grade(b,m1)=1`
`grade(a,db)=1`
`grade(c,db)=h`

`diff(se)=f`
`diff(ds)=t`
`grade(b,se)=1`
`grade(c,se)=1`
`grade(d,ds)=h`
`grade(g,ds)=1`

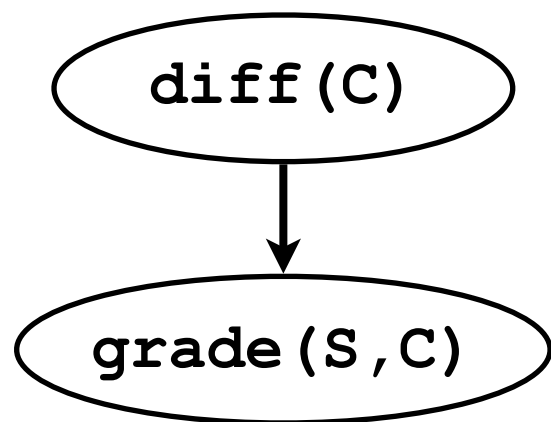
`diff(ai)=t`
`grade(f,ai)=1`
`grade(e,ai)=h`
`grade(g,ai)=h`
`grade(d,ai)=h`
`grade(a,ai)=1`

$$P(\text{grade}=h | \text{diff}=t) = 5/9$$

$$P(\text{grade}=h | \text{diff}=f) =$$

Lifted Graphical Models: Parameter Learning

- **data** = interpretations on ground level
- **estimate** parameters on first-order level taking all instances of same par-RV together



`diff(m1)=t`
`diff(db)=f`
`grade(a,m1)=h`
`grade(b,m1)=1`
`grade(a,db)=1`
`grade(c,db)=h`

`diff(se)=f`
`diff(ds)=t`
`grade(b,se)=1`
`grade(c,se)=1`
`grade(d,ds)=h`
`grade(g,ds)=1`

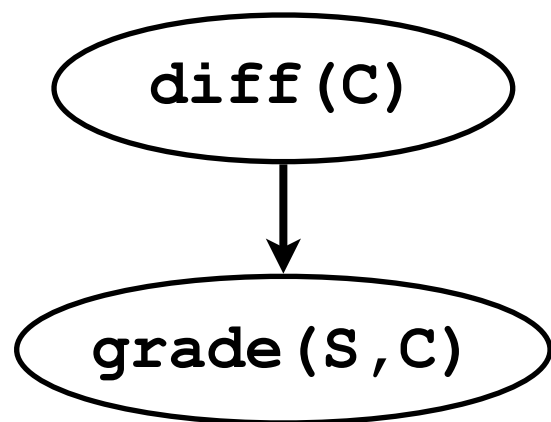
`diff(ai)=t`
`grade(f,ai)=1`
`grade(e,ai)=h`
`grade(g,ai)=h`
`grade(d,ai)=h`
`grade(a,ai)=1`

$$P(\text{grade}=h | \text{diff}=t) = 5/9$$

$$P(\text{grade}=h | \text{diff}=f) = 1/4$$

Lifted Graphical Models: Parameter Learning

- **data** = interpretations on ground level
- **estimate** parameters on first-order level taking all instances of same par-RV together



`diff(m1)=t`
`diff(db)=f`
`grade(a,m1)=h`
`grade(b,m1)=1`
`grade(a,db)=1`
`grade(c,db)=h`

`diff(se)=f`
`diff(ds)=t`
`grade(b,se)=1`
`grade(c,se)=1`
`grade(d,ds)=h`
`grade(g,ds)=1`

`diff(ai)=t`
`grade(f,ai)=1`
`grade(e,ai)=h`
`grade(g,ai)=h`
`grade(d,ai)=h`
`grade(a,ai)=1`

$$P(\text{grade}=h | \text{diff}=t) = 5/9$$

$$P(\text{grade}=h | \text{diff}=f) = 1/4$$

$$P(\text{diff}=t) = 3/5$$

Structure Learning

- often based on search over candidate structures of specific form
- close connection to both inductive logic programming and structure learning in graphical models

Languages to define par-factor graphs

- relational Markov networks (RMNs) [Taskar et al 2002]
- Markov logic networks (MLNs) [Richardson & Domingos 2006]
- probabilistic soft logic (PSL) [Broecheler et al 2010]
- FACTORIE [McCallum et al 2009]
- Bayesian logic programs (BLPs) [Kersting & De Raedt 2001]
- relational Bayesian networks (RBNs) [Jaeger 2002]
- logical Bayesian networks (LBNs) [Fierens et al 2005]
- probabilistic relational models (PRMs) [Koller & Pfeffer 1998]
- Bayesian logic (BLOG) [Milch et al 2005]
- CLP(BN) [Santos Costa et al 2008]
- probabilistic programming languages such as ProbLog, PRISM, Church, ...
- and many more ...

Languages to define par-factor graphs

- relational Markov networks (RMNs) [Taskar et al 2002]
- Markov logic networks (MLNs) [Richardson & Domingos 2006]
- probabilistic soft logic (PSL) [Broecheler et al 2010]
- FACTORIE [McCallum et al 2009]
- Bayesian logic programs (BLPs) [Kersting & De Raedt 2001]
- relational Bayesian networks (RBNs) [Jaeger 2002]
- logical Bayesian networks (LBNs) [Fierens et al 2005]
- probabilistic relational models (PRMs) [Koller & Pfeffer 1998]
- Bayesian logic (BLOG) [Milch et al 2005]
- CLP(BN) [Santos Costa et al 2008]
- probabilistic programming languages such as ProbLog, PRISM, Church, ...
- and many more ...

CLP(BN)

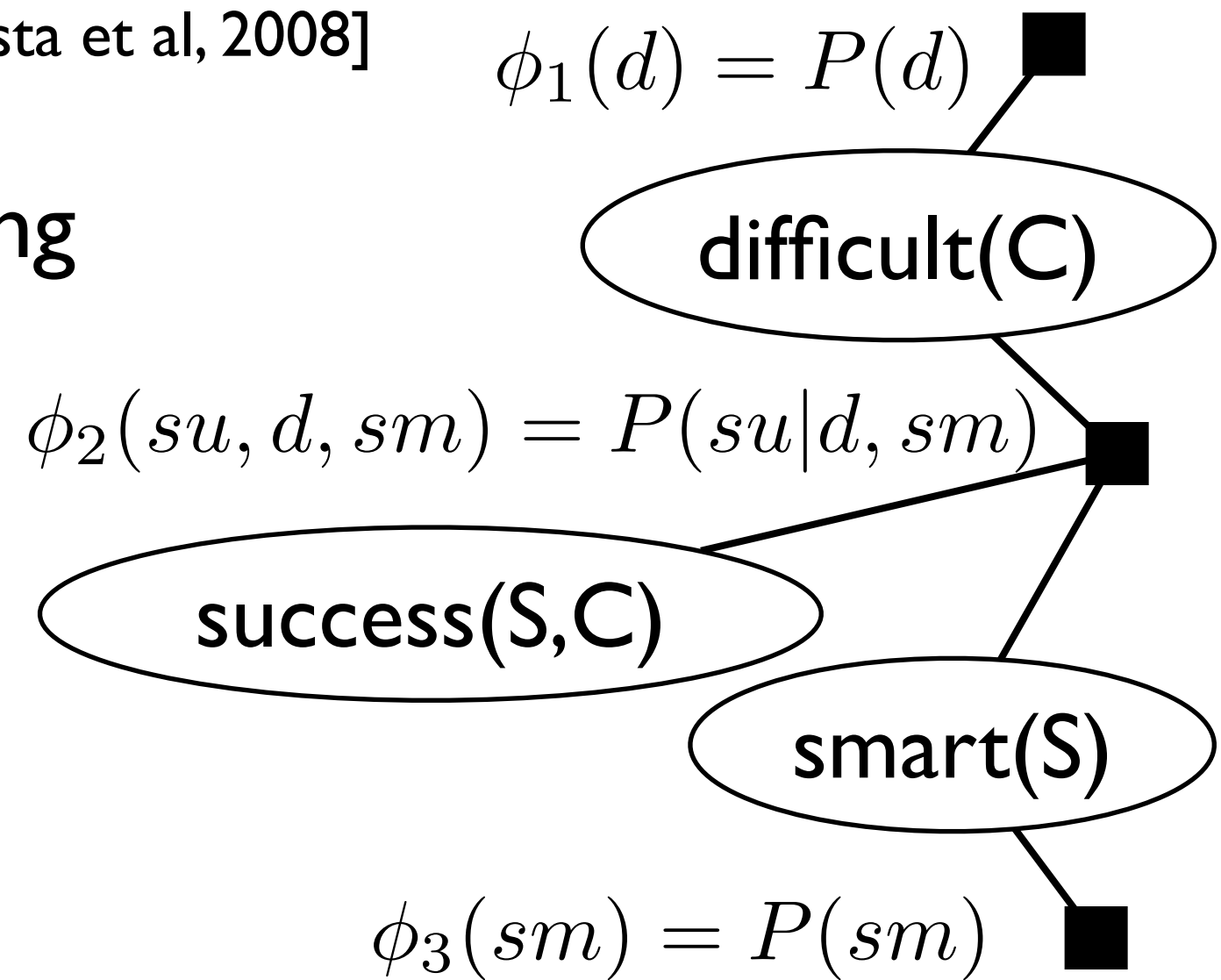
[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks



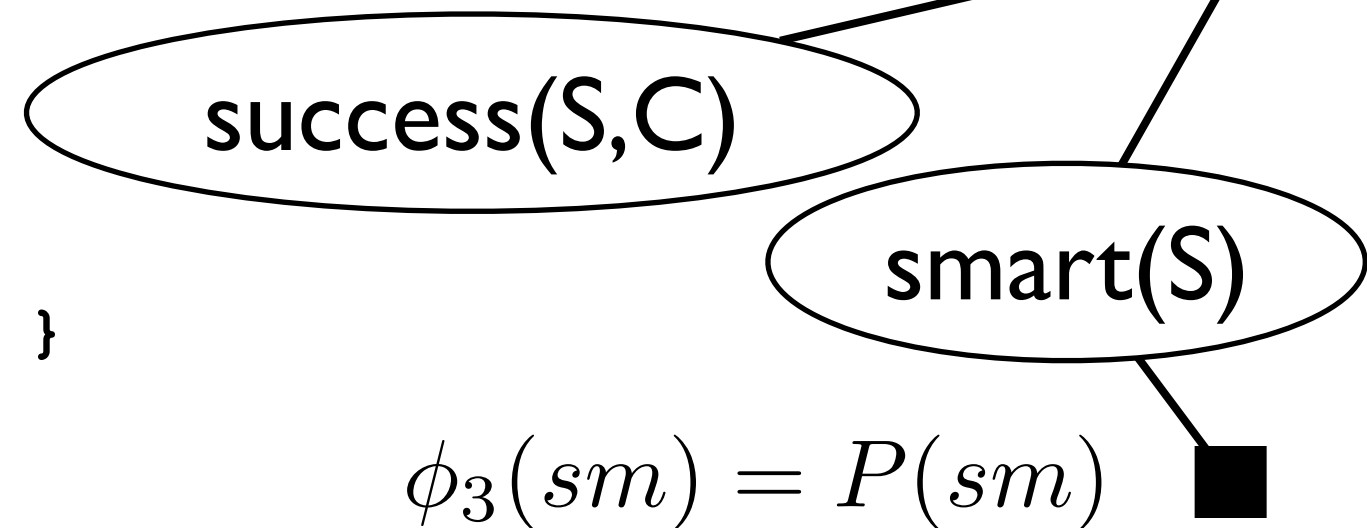
CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks

```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f],[0.6,0.4],[1]) }
```

$$\phi_2(su, d, sm) = P(su|d, sm)$$



CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks

par-RV: unique skolem term

```
difficult(Course, Diff) :-  
  { Diff = d(Course)  
    with p([τ, τ], [0.6, 0.4], []) }  
  }
```

$$\phi_1(d) = P(d)$$

difficult(C)

$$\phi_2(su, d, sm) = P(su|d, sm)$$

success(S,C)

smart(S)

$$\phi_3(sm) = P(sm)$$

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks

par-RV: unique skolem term

```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f], [0.6,0.4], []) }  
possible values
```

$$\phi_1(d) = P(d)$$

difficult(C)

$$\phi_2(su, d, sm) = P(su|d, sm)$$

success(S,C)

smart(S)

$$\phi_3(sm) = P(sm)$$

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks

par-RV: unique skolem term

```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f],[0.6,0.4],[1]) }
```

possible values

their probabilities

$$\phi_1(d) = P(d)$$

difficult(C)

$$\phi_2(su, d, sm) = P(su|d, sm)$$

success(S,C)

smart(S)

$$\phi_3(sm) = P(sm)$$

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks

par-RV: unique skolem term

```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f],[0.6,0.4] [1]) }
```

possible values

their probabilities

list of parents

$$\phi_1(d) = P(d)$$

difficult(C)

$$\phi_2(su, d, sm) = P(su|d, sm)$$

success(S,C)

smart(S)

$$\phi_3(sm) = P(sm)$$

CLP(BN)

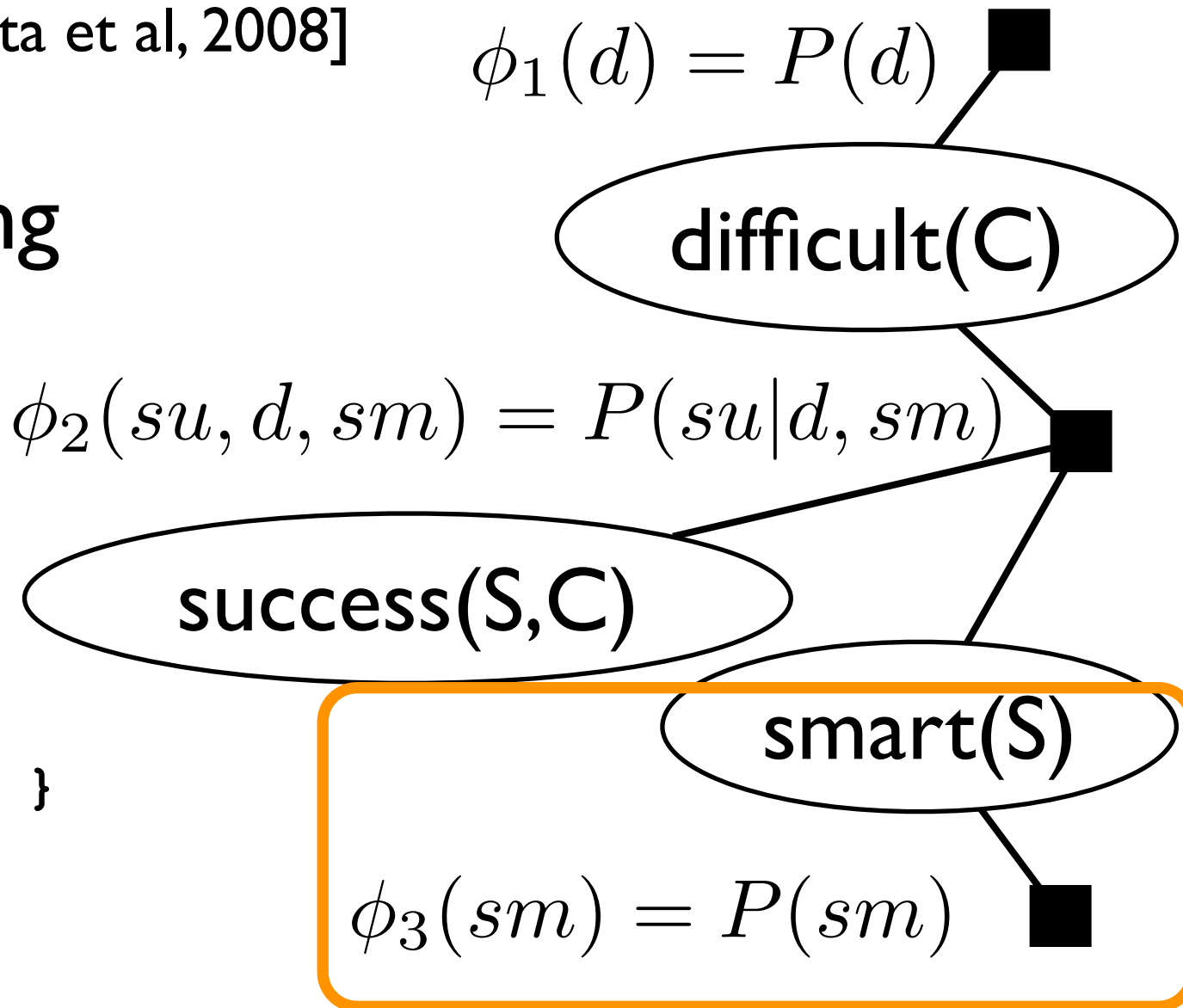
[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks

```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f],[0.6,0.4],[1]) }
```

```
smart(Student,Smart) :-  
  { Smart = sm(Student)  
    with p([t,f],[0.7,0.3],[1]) }
```

$$\phi_2(su, d, sm) = P(su|d, sm)$$



CLP(BN)

[Santos Costa et al, 2008]

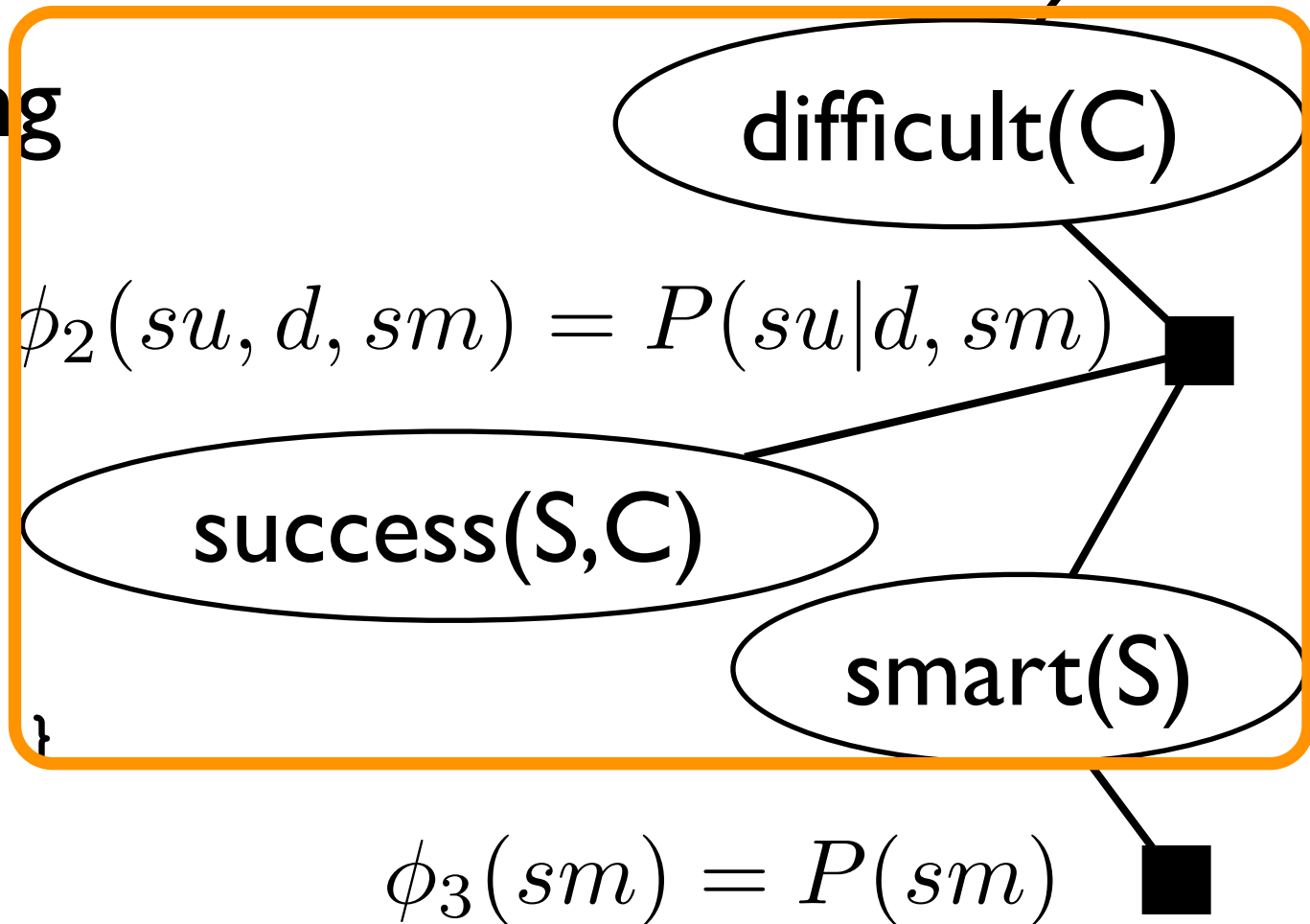
$$\phi_1(d) = P(d)$$

constraint logic programming
for Bayesian networks

```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f],[0.6,0.4],[1]) }
```

```
smart(Student,Smart) :-  
  { Smart = sm(Student)  
    with p([t,f],[0.7,0.3],[1]) }
```

```
success(Student,Course,Success) :-  
  takes(Student,Course),  
  difficult(Course,D),  
  smart(Student,Sm),  
  { Success = su(Student,Course)  
    with p([t,f],[0.85,0.1,0.98,0.45,  
                0.15,0.9,0.02,0.55],[D,Sm]) }
```



CLP(BN)

[Santos Costa et al, 2008]

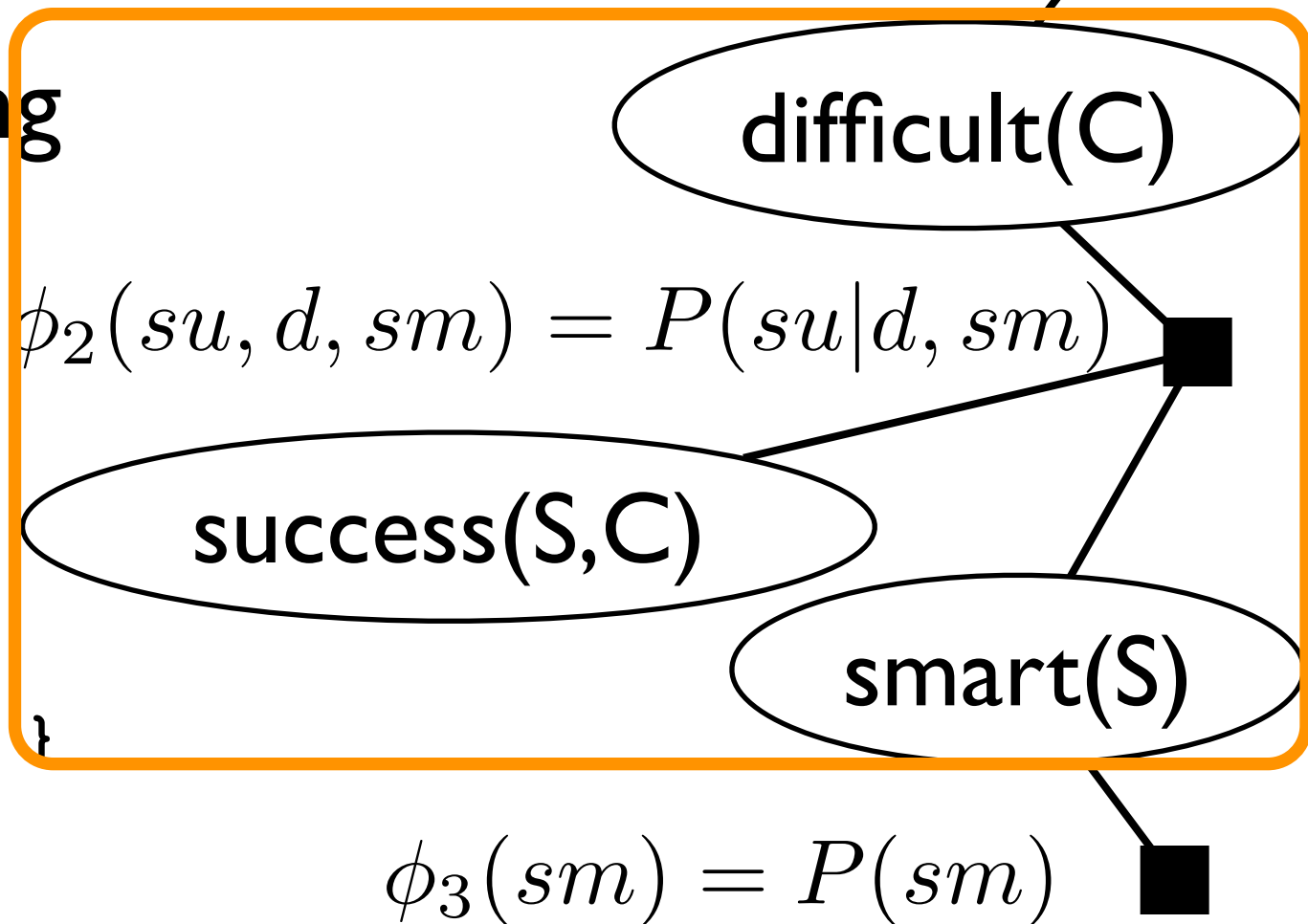
$$\phi_1(d) = P(d)$$

constraint logic programming
for Bayesian networks

```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f],[0.6,0.4],[1]) }
```

```
smart(Student,Smart) :-  
  { Smart = sm(Student)  
    with p([t,f],[0.7,0.3],[1]) }
```

```
success(Student,Course,Success) :-  
  takes(Student,Course),  
  difficult(Course,D),  
  smart(Student,Sm),  
  { Success = su(Student,Course)  
    with p([t,f],[0.85,0.1,0.98,0.45,  
              0.15,0.9,0.02,0.55],[D,Sm]) }
```



Prolog call constrains
grounding

CLP(BN)

[Santos Costa et al, 2008]

$$\phi_1(d) = P(d)$$

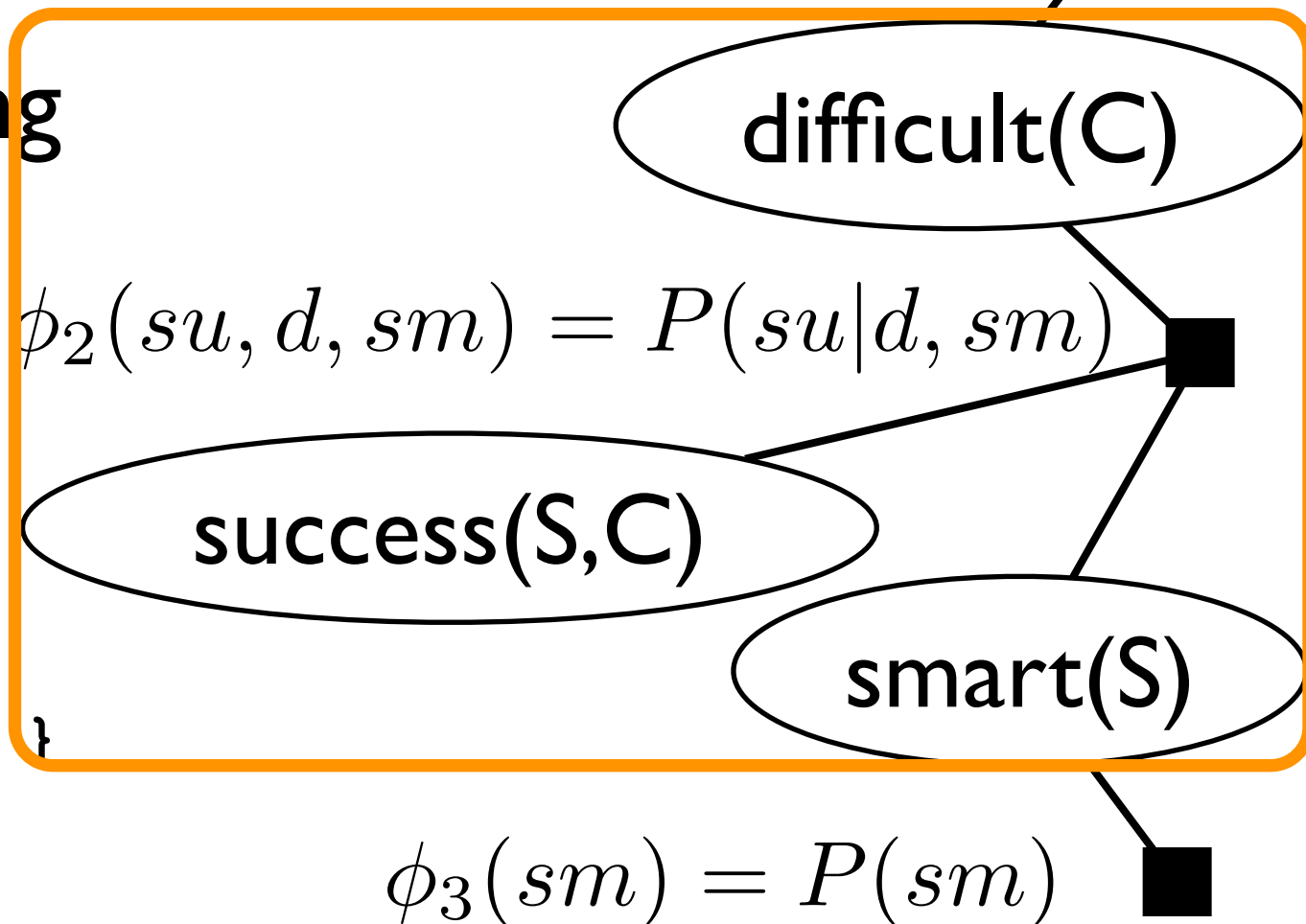
constraint logic programming
for Bayesian networks

```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f],[0.6,0.4],[1]) }
```

```
smart(Student,Smart) :-  
  { Smart = sm(Student)  
    with p([t,f],[0.7,0.3],[1]) }
```

```
success(Student,Course,Success) :-  
  takes(Student,Course),  
  difficult(Course,D),  
  smart(Student,Sm),  
  { Success = su(Student,Course)  
    with p([t,f],[0.85,0.1,0.98,0.45,  
              0.15,0.9,0.02,0.55], [D,Sm]) }
```

get parents



CLP(BN)

[Santos Costa et al, 2008]

$$\phi_1(d) = P(d)$$

constraint logic programming
for Bayesian networks

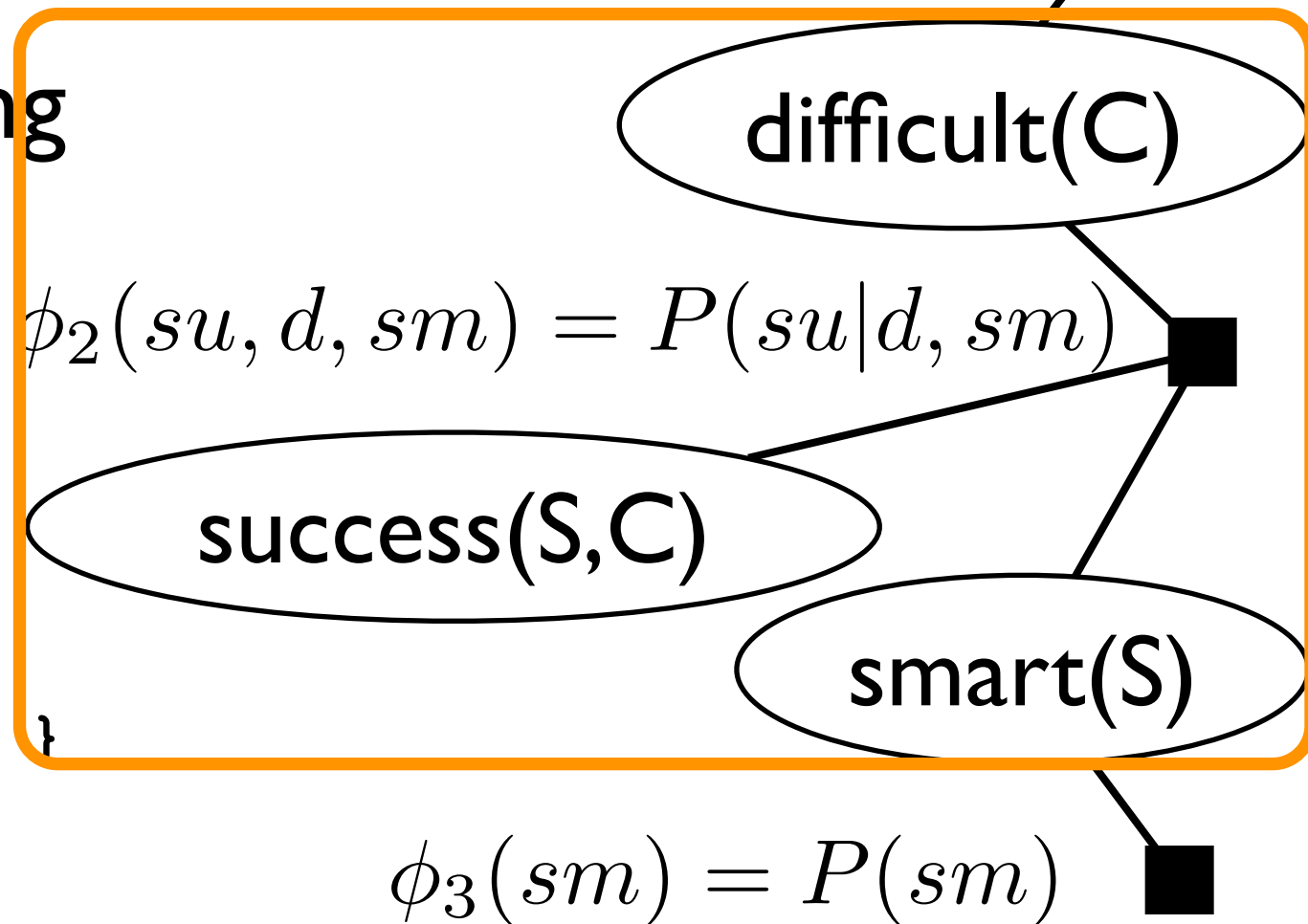
```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f],[0.6,0.4],[1]) }
```

```
smart(Student,Smart) :-  
  { Smart = sm(Student)  
    with p([t,f],[0.7,0.3],[1]) }
```

```
success(Student,Course,Success) :-  
  takes(Student,Course),  
  difficult(Course,D),  
  smart(Student,Sm),  
  { Success = su(Student,Course)  
    with p([t,f],[0.85,0.1,0.98,0.45,  
               0.15,0.9,0.02,0.55], [D,Sm]) }
```

$$P(su|d=t,sm=t)$$

get parents



CLP(BN)

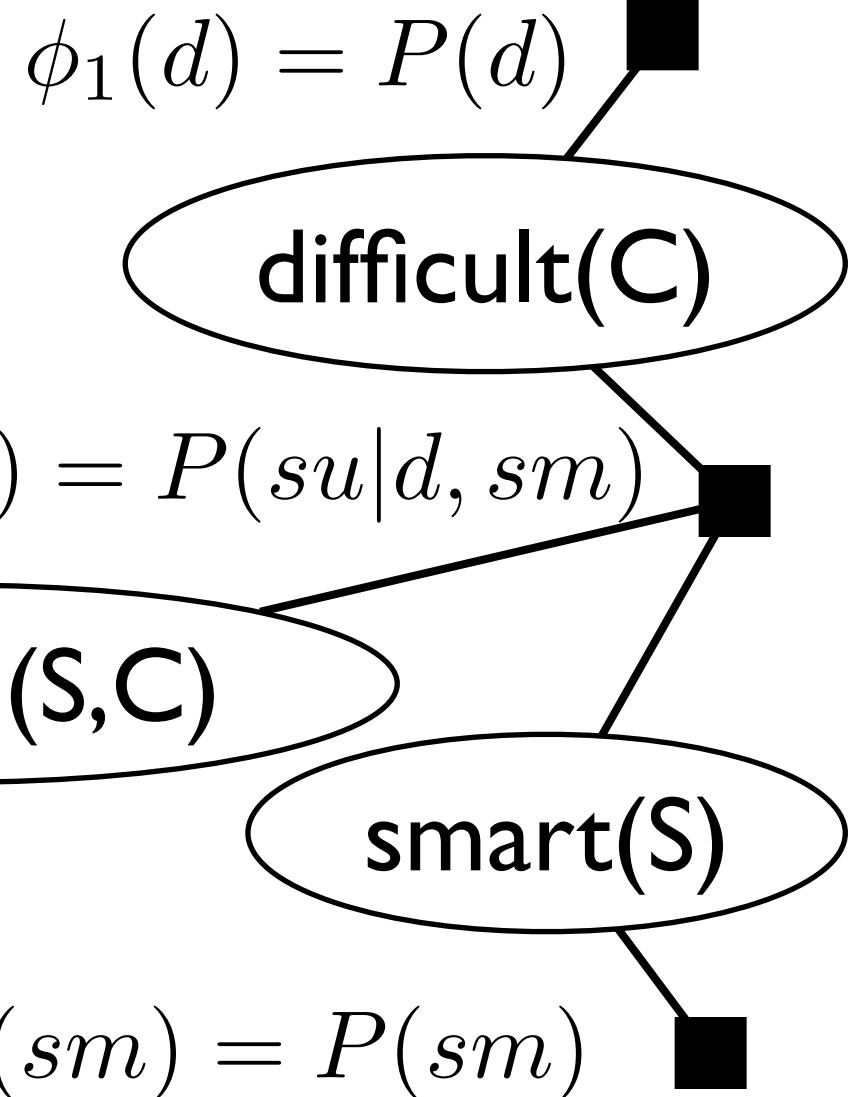
[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks

```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f],[0.6,0.4],[ ]) }
```

```
smart(Student,Smart) :-  
  { Smart = sm(Student)  
    with p([t,f],[0.7,0.3],[ ]) }
```

```
success(Student,Course,Success) :-  
  takes(Student,Course) ,  
  difficult(Course,D) ,  
  smart(Student,Sm) ,  
  { Success = su(Student,Course)  
    with p([t,f],[0.85,0.1,0.98,0.45,  
               0.15,0.9,0.02,0.55],[D,Sm]) }
```



CLP(BN)

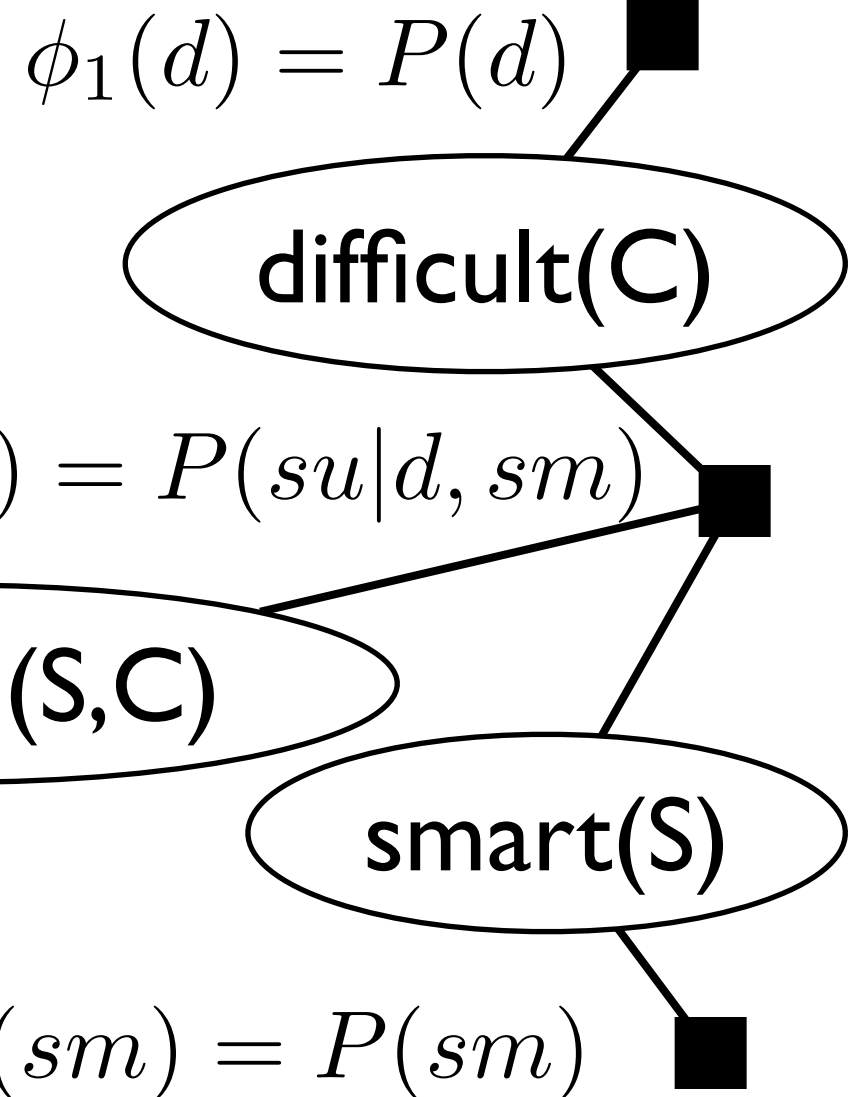
[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks

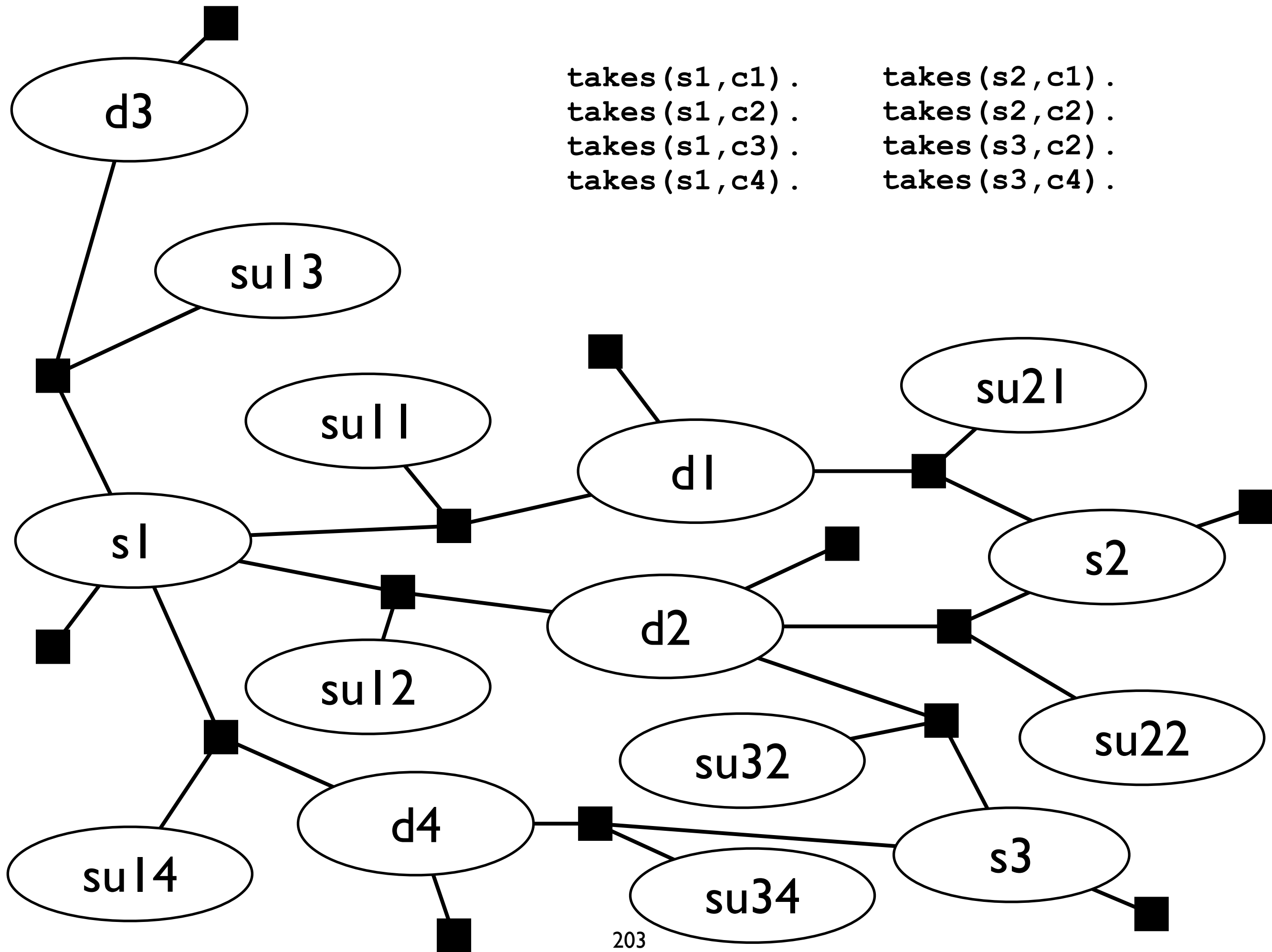
```
difficult(Course,Diff) :-  
  { Diff = d(Course)  
    with p([t,f],[0.6,0.4],[1]) }
```

```
smart(Student,Smart) :-  
  { Smart = sm(Student)  
    with p([t,f],[0.7,0.3],[1]) }
```

```
success(Student,Course,Success) :-  
  takes(Student,Course),  
  difficult(Course,D),  
  smart(Student,Sm),  
  { Success = su(Student,Course)  
    with p([t,f],[0.85,0.1,0.98,0.45,  
               0.15,0.9,0.02,0.55],[D,Sm]) }
```



```
takes(s1,c1).  
takes(s1,c2).  
takes(s1,c3).  
takes(s1,c4).  
takes(s2,c1).  
takes(s2,c2).  
takes(s3,c2).  
takes(s3,c4).
```



in ProbLog?

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
  
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
  
success(Student,Course,Success) :-  
    takes(Student,Course) ,  
    difficult(Course,D) ,  
    smart(Student,Sm) ,  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45,  
                  0.15,0.9,0.02,0.55] , [D,Sm]) }
```

in ProbLog?

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[ ]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[ ]) }

success(Student,Course,Success) :-
    takes(Student,Course) ,
    difficult(Course,D) ,
    smart(Student,Sm) ,
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                  0.15,0.9,0.02,0.55],[D,Sm]) }
```

```
0.6::difficult(C) :- takes(_,C) .
0.7::smart(S) :- takes(S,_).
0.85::success(S,C) <- takes(S,C) , difficult(C) , smart(S) .
0.10::success(S,C) <- takes(S,C) , difficult(C) , \+smart(S) .
0.98::success(S,C) <- takes(S,C) , \+difficult(C) , smart(S) .
0.45::success(S,C) <- takes(S,C) , \+difficult(C) , \+smart(S) .
```

Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
  
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
  
success(Student,Course,Success) :-  
    takes(Student,Course) ,  
    difficult(Course,D) ,  
    smart(Student,Sm) ,  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                  0.15,0.9,0.02,0.55],[D,Sm]) }  
}
```

```
takes(s1,c1) .  
takes(s1,c2) .  
takes(s2,c1) .  
takes(s2,c2) .  
takes(s3,c2) .  
takes(s3,c4) .
```

Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
success(Student, Course, Success) :-  
    takes(Student, Course) ,  
    difficult(Course, D) ,  
    smart(Student, Sm) ,  
    { Success = su(Student, Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                0.15,0.9,0.02,0.55],[D,Sm]) }
```

takes(s1,c1) .

takes(s1,c2) .

takes(s2,c1) .

takes(s2,c2) .

takes(s3,c2) .

takes(s3,c4) .

?-success(s2,c1,S) .



Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
  
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
  
success(Student,Course,Success) :-  
    takes(Student,Course),  
    difficult(Course,D),  
    smart(Student,Sm),  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                  0.15,0.9,0.02,0.55],[D,Sm]) }  
  
?-success(s2,c1,S).
```

```
takes(s1,c1).  
takes(s1,c2).  
takes(s2,c1).  
takes(s2,c2).  
takes(s3,c2).  
takes(s3,c4).
```



Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
}
```

```
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
}
```

```
success(Student,Course,Success) :-  
    takes(Student,Course),  
    difficult(Course,D)  
    smart(Student,Sm),  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                  0.15,0.9,0.02,0.55],[D,Sm]) }  
}
```

```
takes(s1,c1).  
takes(s1,c2).  
takes(s2,c1).  
takes(s2,c2).  
takes(s3,c2).  
takes(s3,c4).
```

```
?-success(s2,c1,S).
```



Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
}
```

```
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
}
```

```
success(Student,Course,Success) :-  
    takes(Student,Course),  
    difficult(Course,D)  
    smart(Student,Sm),  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                  0.15,0.9,0.02,0.55],[D,Sm]) }  
}
```

```
takes(s1,c1).  
takes(s1,c2).  
takes(s2,c1).  
takes(s2,c2).  
takes(s3,c2).  
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
success(Student,Course,Success) :-  
    takes(Student,Course),  
    difficult(Course,D),  
    smart(Student,Sm),  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                0.15,0.9,0.02,0.55],[D,Sm]) }
```

```
takes(s1,c1).  
takes(s1,c2).  
takes(s2,c1).  
takes(s2,c2).  
takes(s3,c2).  
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
success(Student,Course,Success) :-  
    takes(Student,Course),  
    difficult(Course,D),  
    smart(Student,Sm),  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                0.15,0.9,0.02,0.55],[D,Sm]) }
```

```
takes(s1,c1).  
takes(s1,c2).  
takes(s2,c1).  
takes(s2,c2).  
takes(s3,c2).  
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

Sm=sm(s2) with p(...,[])

D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
  
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
  
success(Student,Course,Success) :-  
    takes(Student,Course) ,  
    difficult(Course,D) ,  
    smart(Student,Sm) ,  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                  0.15,0.9,0.02,0.55],[D,Sm]) }  
}
```

```
takes(s1,c1) .  
takes(s1,c2) .  
takes(s2,c1) .  
takes(s2,c2) .  
takes(s3,c2) .  
takes(s3,c4) .
```

```
?-success(s2,c1,S) .
```

Sm=sm(s2) with p(...,[])
D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
  
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
  
success(Student,Course,Success) :-  
    takes(Student,Course) ,  
    difficult(Course,D) ,  
    smart(Student,Sm) ,  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                  0.15,0.9,0.02,0.55] , [D,Sm]) }  
}
```

```
takes(s1,c1) .  
takes(s1,c2) .  
takes(s2,c1) .  
takes(s2,c2) .  
takes(s3,c2) .  
takes(s3,c4) .
```

?-success(s2,c1,S) .

Success=su(s2,c1) with p(...,[D,Sm])

Sm=sm(s2) with p(...,[])

D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
  
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
  
success(Student,Course,Success) :-  
    takes(Student,Course),  
    difficult(Course,D),  
    smart(Student,Sm),  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                  0.15,0.9,0.02,0.55],[D,Sm]) }
```

```
takes(s1,c1).  
takes(s1,c2).  
takes(s2,c1).  
takes(s2,c2).  
takes(s3,c2).  
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

Success=su(s2,c1) with p(...,[D,Sm])

Sm=sm(s2) with p(...,[])

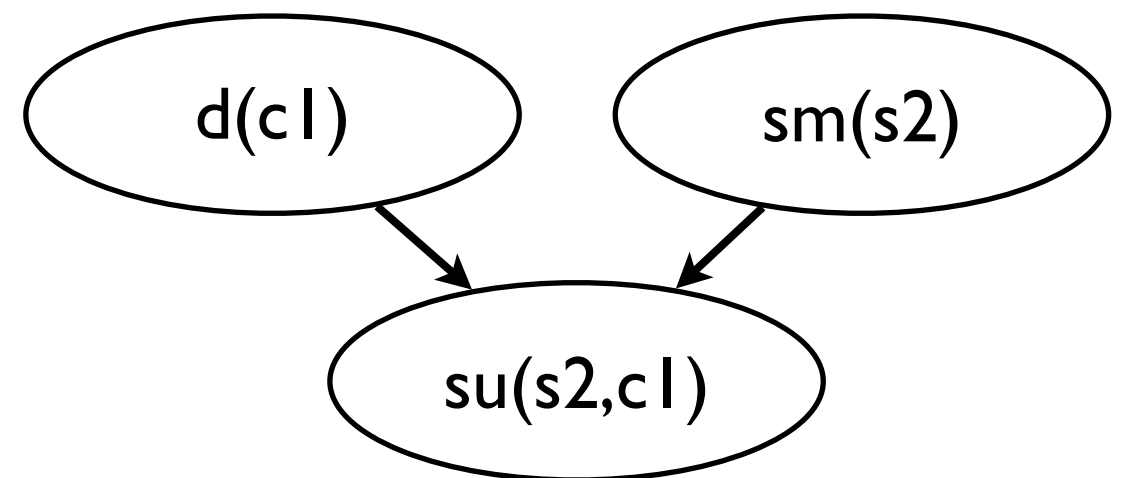
D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-  
    { Diff = d(Course)  
      with p([t,f],[0.6,0.4],[ ]) }  
  
smart(Student,Smart) :-  
    { Smart = sm(Student)  
      with p([t,f],[0.7,0.3],[ ]) }  
  
success(Student,Course,Success) :-  
    takes(Student,Course),  
    difficult(Course,D),  
    smart(Student,Sm),  
    { Success = su(Student,Course)  
      with p([t,f],[0.85,0.1,0.98,0.45  
                  0.15,0.9,0.02,0.55],[D,Sm]) }  
  
?-success(s2,c1,S).
```

```
takes(s1,c1).  
takes(s1,c2).  
takes(s2,c1).  
takes(s2,c2).  
takes(s3,c2).  
takes(s3,c4).
```

Success=su(s2,c1) with p(...,[D,Sm])
Sm=sm(s2) with p(...,[])
D=d(c1) with p(...,[])



CLP(BN) Summary

- Templating Bayesian networks via constraint logic programming
- Knowledge-based model construction (KBMC):
 - construct relevant ground BN by backward reasoning, adding constraints to constraint store
 - run any propositional inference technique

Lifted Graphical Models

Summary

- graphical model + relational templating language
- par-factor graphs as unifying framework
- aggregation / combining rules
- grounding vs lifted inference
- many different languages
 - CLP(BN): directed models via constraint LP

Roadmap

- Modeling
- Reasoning
- Language extensions
- Advanced topics

... with some detours on the way

A key question in AI:



Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

A key question in AI:

Dealing with uncertainty

- probability theory

models

- Our answer: probabilistic logic programming
= probabilistic choices + logic program
- Many languages, systems, applications, ...
- ... and much more to do!

Statistical relational learning, probabilistic logic
learning, probabilistic programming, ...

Maurice Bruynooghe

Bart Demoen

Luc De Raedt

Anton Dries

Daan Fierens

Jason Filippou

Bernd Gutmann

Manfred Jaeger

Gerda Janssens

Kristian Kersting

Theofrastos Mantadelis

Wannes Meert

Bogdan Moldovan

Siegfried Nijssen

Davide Nitti

Joris Renkens

Kate Revoredo

Ricardo Rocha

Vitor Santos Costa

Dimitar Shterionov

Ingo Thon

Hannu Toivonen

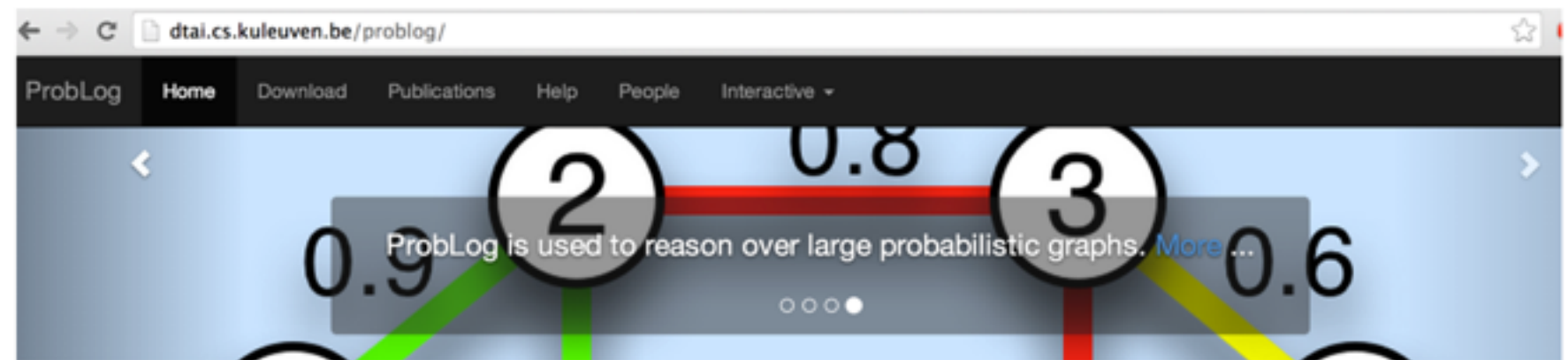
Guy Van den Broeck

Mathias Verbeke

Jonas Vlasselaer

Thanks !

<http://dtai.cs.kuleuven.be/problog>



Introduction.

Probabilistic logic programs are logic programs in which some of the facts are annotated with probabilities.

ProbLog is a tool that allows you to intuitively build programs that do not only encode **complex interactions** between a large sets of **heterogenous components** but also the inherent **uncertainties** that are present in real-life situations.

The engine tackles several tasks such as computing the marginals given evidence and learning from (partial) interpretations. ProbLog is a suite of efficient algorithms for various inference tasks. It is based on a conversion of the program and the queries and evidence to a weighted Boolean formula. This allows us to reduce the inference tasks to well-studied tasks such as weighted model counting, which can be solved using state-of-the-art methods known from the graphical model and knowledge compilation literature.

The Language. Probabilistic Logic Programming.

ProbLog makes it easy to express complex, probabilistic models.

```
0.3::stress(X) :- person(X).
0.2::influences(X,Y) :- person(X), person(Y).
```

PLP Systems

- **PRISM** <http://sato-www.cs.titech.ac.jp/prism/>
- **ProbLog2** <http://dtai.cs.kuleuven.be/problog/>
- Yap Prolog <http://www.dcc.fc.up.pt/~vsc/Yap/> includes
 - **ProbLog I**
 - **cplint** <https://sites.google.com/a/unife.it/ml/cplint>
 - **CLP(BN)**
 - **LP2**
- **PITA** in XSB Prolog <http://xsb.sourceforge.net/>
- **AILog2** <http://artint.info/code/ailog/ailog2.html>
- **SLPs** <http://stoics.org.uk/~nicos/sware/pepl>
- **contdist** <http://www.cs.sunysb.edu/~cram/contdist/>
- **DC** <https://code.google.com/p/distributional-clauses>
- **WFOMC** <http://dtai.cs.kuleuven.be/ml/systems/wfomc>

References

- Bellodi E, Riguzzi F (2011) Learning the structure of probabilistic logic programs. In: Proceedings of the 21st International Conference on Inductive Logic Programming (ILP-11)
- Bellodi E, Riguzzi F (2014a) Expectation maximization over binary decision diagrams for probabilistic logic programs. *Intelligent Data Analysis* 17(2):343–363
- Bellodi E, Riguzzi F (2014b) Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming (TPLP)* FirstView
- Bellodi E, Lamma E, Riguzzi F, Santos Costa V, Zese R (2014) Lifted probabilistic logic programming. In: Proceedings of the 30th International Conference on Logic Programming (ICLP-14)
- Broecheler M, Mihalkova L, Getoor L (2010) Probabilistic similarity logic. In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI-10)
- Bryant RE (1986) Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691
- Cohen SB, Simmons RJ, Smith NA (2008) Dynamic programming algorithms as products of weighted logic programs. In: Proceedings of the 24th International Conference on Logic Programming (ICLP-08)
- Cussens J (2001) Parameter estimation in stochastic logic programs. *Machine Learning* 44(3):245–271
- De Maeyer D, Renkens J, Cloots L, De Raedt L, Marchal K (2013) Phenetic: network-based interpretation of unstructured gene lists in *e. coli*. *Molecular BioSystems* 9(7):1594–1603
- De Raedt L, Kimmig A (2013) Probabilistic programming concepts. CoRR abs/1312.4328
- De Raedt L, Thon I (2010) Probabilistic rule learning. In: Revised Papers of the 20th International Conference on Inductive Logic Programming (ILP-10)
- De Raedt L, Kimmig A, Toivonen H (2007) ProbLog: A probabilistic Prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)
- De Raedt L, Frasconi P, Kersting K, Muggleton S (eds) (2008) Probabilistic Inductive Logic Programming — Theory and Applications, Lecture Notes in Artificial Intelligence, vol 4911. Springer
- De Raedt L, Kersting K, Kimmig A, Revoredo K, Toivonen H (2008) Compressing probabilistic Prolog programs. *Machine Learning* 70(2-3):151–168
- Eisner J, Goldlust E, Smith N (2005) Compiling Comp Ling: Weighted dynamic programming and the Dyna language. In: Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP-05)
- Fierens D, Blockeel H, Bruynooghe M, Ramon J (2005) Logical Bayesian networks and their relation to other probabilistic logical models. In: Proceedings of the 15th International Conference on Inductive Logic Programming (ILP-05)
- Fierens D, Van den Broeck G, Bruynooghe M, De Raedt L (2012) Constraints for probabilistic logic programming. In: Proceedings of the NIPS Probabilistic Programming Workshop

- Fierens D, Van den Broeck G, Renkens J, Shterionov D, Gutmann B, Thon I, Janssens G, De Raedt L (2014) Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming (TPLP)* FirstView
- Getoor L, Friedman N, Koller D, Pfeffer A, Taskar B (2007) Probabilistic relational models. In: Getoor L, Taskar B (eds) *An Introduction to Statistical Relational Learning*, MIT Press, pp 129–174
- Goodman N, Mansinghka VK, Roy DM, Bonawitz K, Tenenbaum JB (2008) Church: a language for generative models. In: Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI-08)
- Gutmann B, Thon I, De Raedt L (2011a) Learning the parameters of probabilistic logic programs from interpretations. In: Proceedings of the 22nd European Conference on Machine Learning (ECML-11)
- Gutmann B, Thon I, Kimmig A, Bruynooghe M, De Raedt L (2011b) The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming (TPLP)* 11(4–5):663–680
- Jaeger M (2002) Relational Bayesian networks: A survey. *Linköping Electronic Articles in Computer and Information Science* 7(015)
- Kersting K, De Raedt L (2001) Bayesian logic programs. CoRR cs.AI/0111058
- Kimmig A, Van den Broeck G, De Raedt L (2011a) An algebraic Prolog for reasoning about possible worlds. In: Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)
- Kimmig A, Demoen B, De Raedt L, Santos Costa V, Rocha R (2011b) On the implementation of the probabilistic logic programming language ProbLog. *Theory and Practice of Logic Programming (TPLP)* 11:235–262
- Koller D, Pfeffer A (1998) Probabilistic frame-based systems. In: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)
- McCallum A, Schultz K, Singh S (2009) FACTORIE: Probabilistic programming via imperatively defined factor graphs. In: Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS-09)
- Milch B, Marthi B, Russell SJ, Sontag D, Ong DL, Kolobov A (2005) Blog: Probabilistic models with unknown objects. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)
- Moldovan B, De Raedt L (2014) Occluded object search by relational affordances. In: IEEE International Conference on Robotics and Automation (ICRA-14)
- Moldovan B, Moreno P, van Otterlo M, Santos-Victor J, De Raedt L (2012) Learning relational affordance models for robots in multi-object manipulation tasks. In: IEEE International Conference on Robotics and Automation (ICRA-12)
- Moldovan B, Thon I, Davis J, De Raedt L (2013) MCMC estimation of conditional probabilities in probabilistic programming languages. In: Proceedings of the 12th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-13)
- Muggleton S (1996) Stochastic logic programs. In: De Raedt L (ed) *Advances in Inductive Logic Programming*, IOS Press, pp 254–264
- Nitti D, De Laet T, De Raedt L (2013) A particle filter for hybrid relational domains. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-13)
- Nitti D, De Laet T, De Raedt L (2014) Relational object tracking and learning. In: IEEE International Conference on Robotics and Automation (ICRA), June

2014

- Pfeffer A (2001) IBAL: A probabilistic rational programming language. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)
- Pfeffer A (2009) Figaro: An object-oriented probabilistic programming language. Tech. rep., Charles River Analytics
- Poole D (2003) First-order probabilistic inference. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)
- Richardson M, Domingos P (2006) Markov logic networks. *Machine Learning* 62(1-2):107–136
- Santos Costa V, Page D, Cussens J (2008) CLP(*BN*): Constraint logic programming for probabilistic knowledge. In: De Raedt et al (2008), pp 156–188
- Sato T (1995) A statistical learning method for logic programs with distribution semantics. In: Proceedings of the 12th International Conference on Logic Programming (ICLP-95)
- Sato T (2011) A general MCMC method for Bayesian inference in logic-based probabilistic modeling. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)
- Sato T, Kameya Y (2001) Parameter learning of logic programs for symbolic-statistical modeling. *J Artif Intell Res (JAIR)* 15:391–454
- Sato T, Kameya Y (2008) New advances in logic-based probabilistic modeling by prism. In: Probabilistic Inductive Logic Programming, pp 118–155
- Skarlatidis A, Artikis A, Filiopou J, Paliouras G (2014) A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming (TPLP)* FirstView
- Suciu D, Olteanu D, Ré C, Koch C (2011) Probabilistic Databases. Synthesis Lectures on Data Management, Morgan & Claypool Publishers
- Taskar B, Abbeel P, Koller D (2002) Discriminative probabilistic models for relational data. In: Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI-02)
- Thon I, Landwehr N, De Raedt L (2008) A simple model for sequences of relational state descriptions. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD-08)
- Thon I, Landwehr N, De Raedt L (2011) Stochastic relational processes: Efficient inference and applications. *Machine Learning* 82(2):239–272
- Van den Broeck G (2011) On the completeness of first-order knowledge compilation for lifted probabilistic inference. In: Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS-11)
- Van den Broeck G, Thon I, van Otterlo M, De Raedt L (2010) DTProbLog: A decision-theoretic probabilistic Prolog. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)
- Van den Broeck G, Taghipour N, Meert W, Davis J, De Raedt L (2011) Lifted probabilistic inference by first-order knowledge compilation. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)
- Van den Broeck G, Meert W, Darwiche A (2014) Skolemization for weighted first-order model counting. In: Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR-14)
- Vennekens J, Verbaeten S, Bruynooghe M (2004) Logic programs with annotated disjunctions. In: Proceedings of the 20th International Conference on Logic Pro-

gramming (ICLP-04)

- Vennekens J, Denecker M, Bruynooghe M (2009) CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming (TPLP)* 9(3):245–308
- Wang WY, Mazaitis K, Cohen WW (2013) Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In: Proceedings of the 22nd ACM International Conference on Information and Knowledge Management (CIKM-13)